LINFO 1121

DATA STRUCTURES AND ALGORITHMS

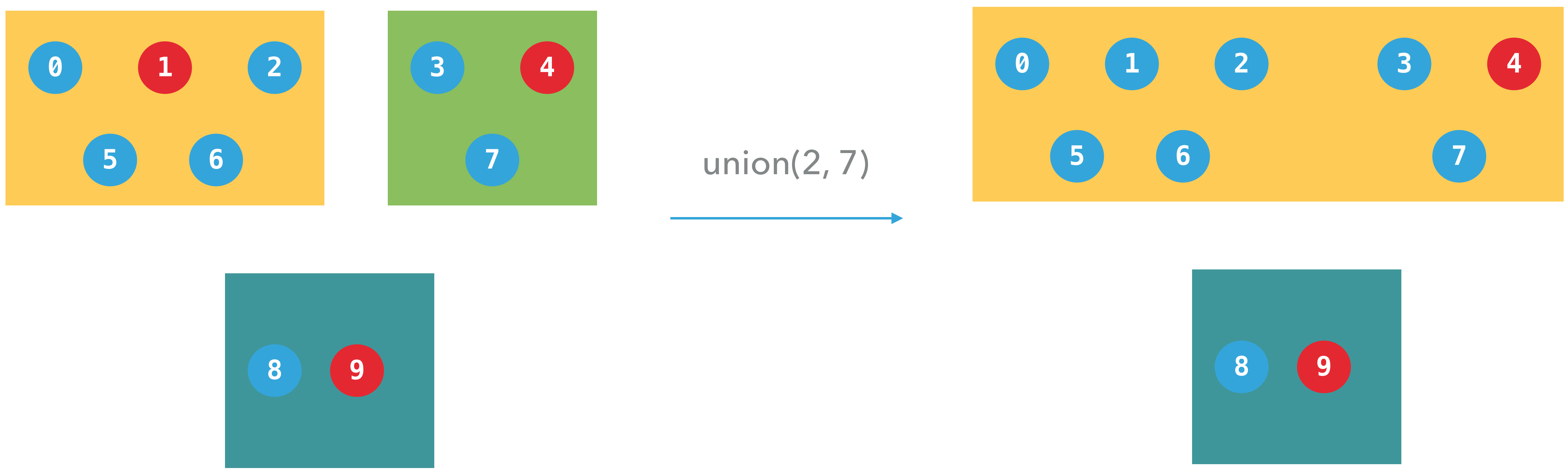
UCLouvain

TP5: PQ, UNION-FIND et HUFFMAN

Question 5.1.1 UNION-FIND

Union-find est une structure de donnée qui, étant donné n éléments, les assigne chacun à un ensemble.

- find(a) retourne le représentant de l'ensemble auquel appartient a
- union(a, b) fait l'union des ensembles auxquels appartiennent a et b.

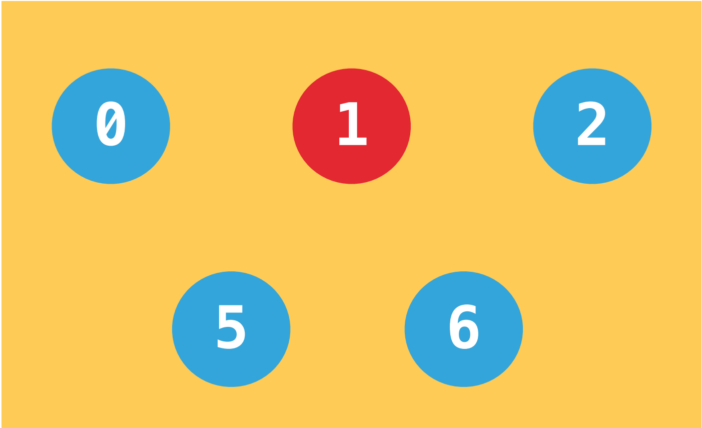


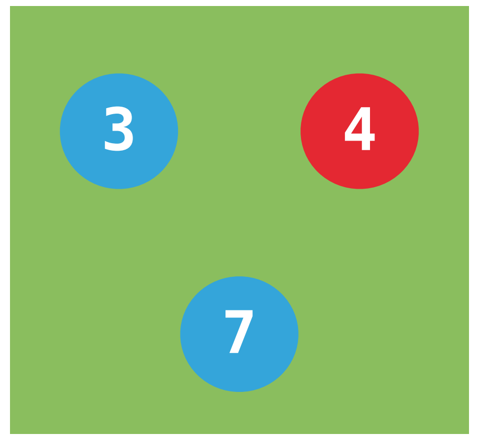
find(0) = 1	find(1) = 1		find(0) = 4	find(1) = 4
find(5) = 1	find(3) = 4		find(5) = 4	find(3) = 4
find(9) = 9			find(9) = 9	

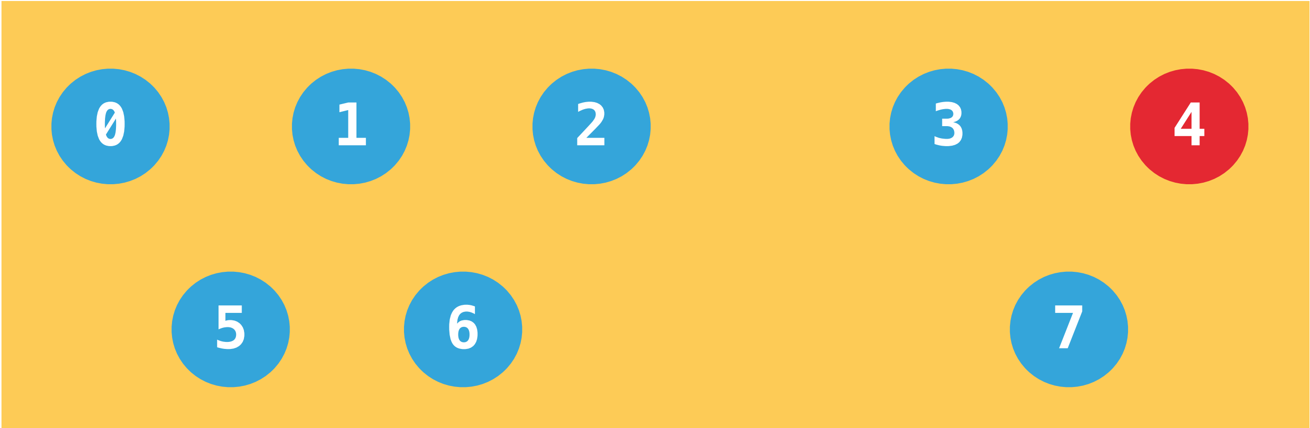
Question 5.1.1 UNION-FIND: QUICK-FIND

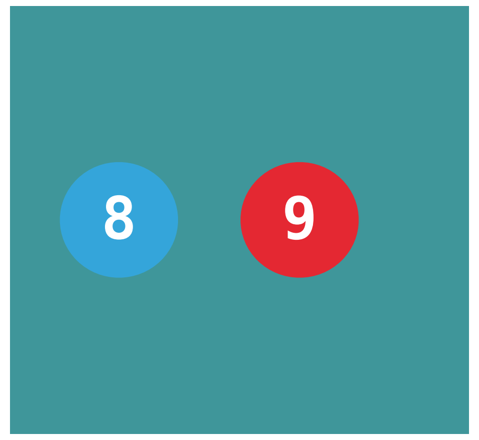
Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	1	4	4	1	1	4	9	9

Val	0	1	2	3	4	5	6	7	8	9
Rep	4	4	4	4	4	4	4	4	9	9

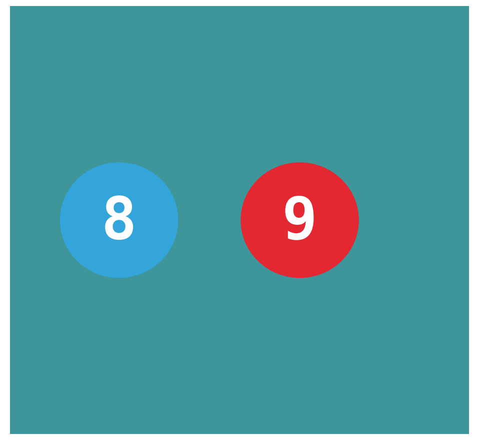






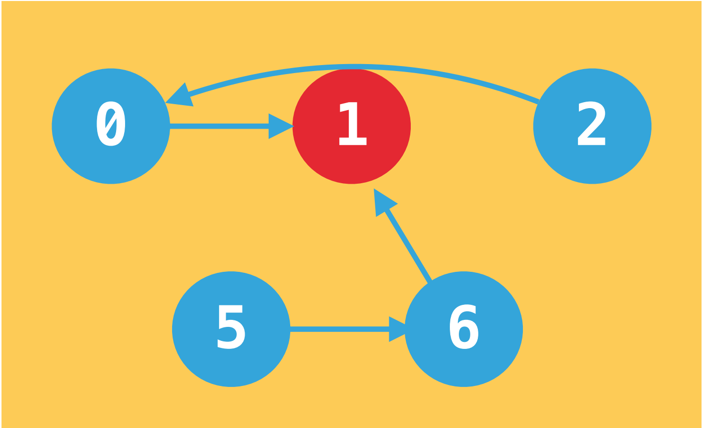


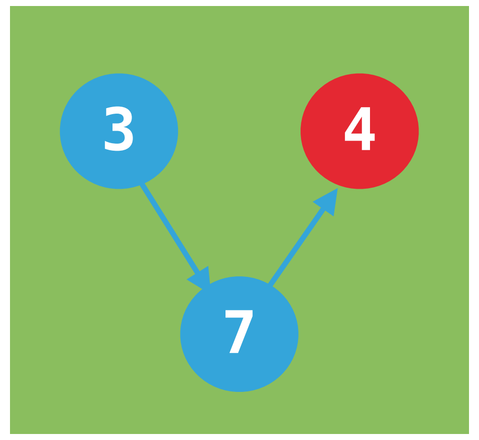

```
int find(int a) { //O(1)
    return tab[a];
}
void union(int a, int b) {
    //O(n)
    a=find(a); b=find(b);
    for(int i = 0; i < n; i++)
        if(tab[i] == a)
            tab[i] = b;
}
```

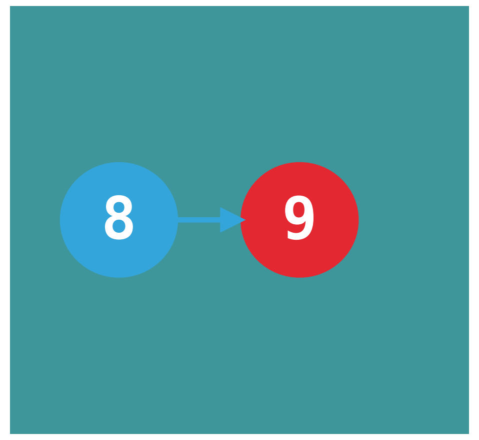


Question 5.1.1 UNION-FIND: QUICK-UNION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	0	7	4	6	1	4	9	9



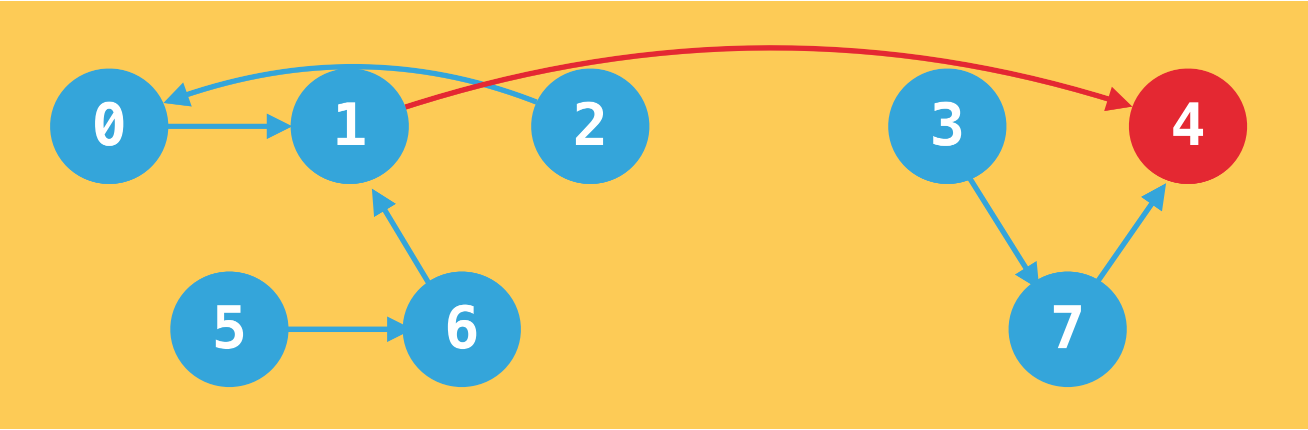


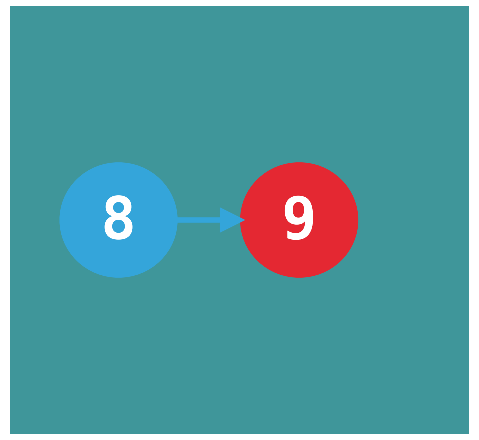



```
int find(int a) { //O(n)
    if(tab[a] != a)
        return find(tab[a]);
    return a;
}

void union(int a, int b) { //O(1) ?
    int i = find(a);
    int j = find(b);
    tab[i] = j;
}
```

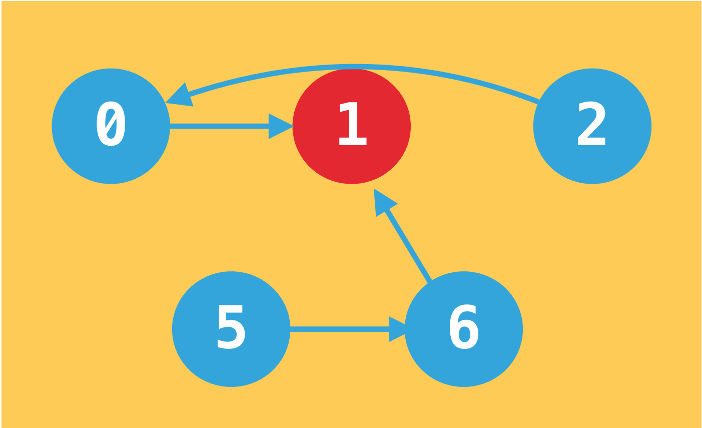
Val	0	1	2	3	4	5	6	7	8	9
Rep	1	4	0	7	4	6	1	4	9	9

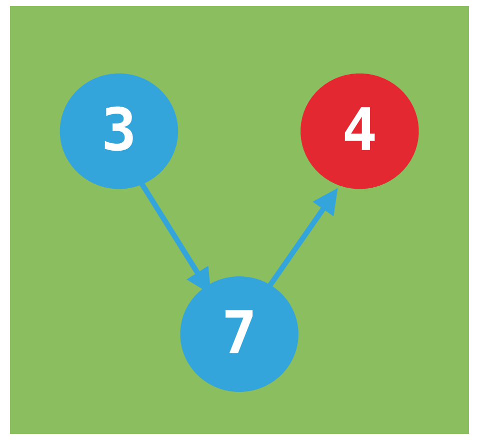


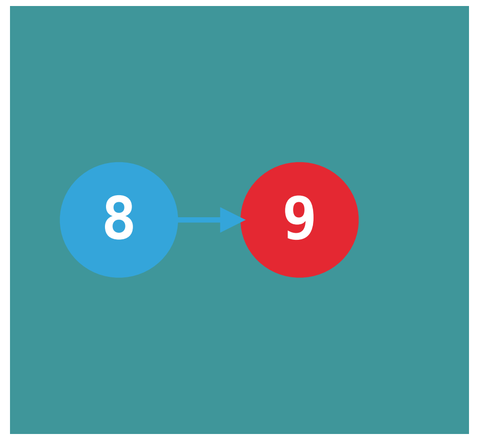


Question 5.1.1 UNION-FIND: WEIGHTED QUICK-UNION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	0	7	4	6	1	4	9	9

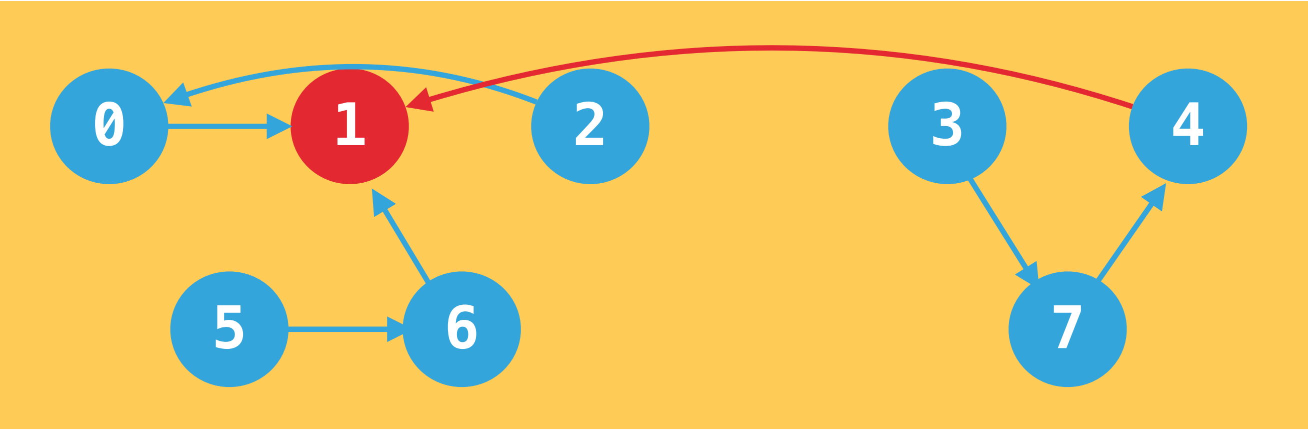


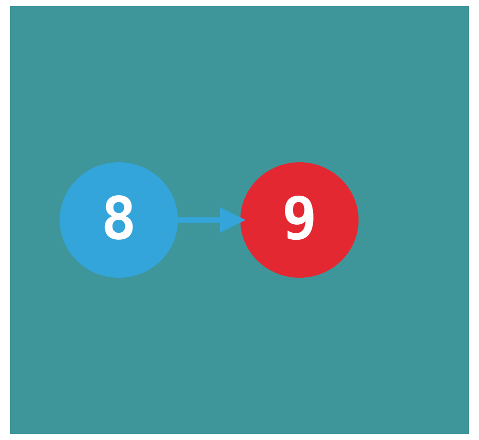





```
int find(int a) { //O(n) ?
    if(tab[a] != a)
        return find(tab[a]);
    return a;
}
void union(int a, int b) { //O(n) ?
    int i = find(a);
    int j = find(b);
    if(size[i] > size[j])
        return union(j, i);
    tab[i] = j;
    size[j] += size[i];
}
```

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	0	7	1	6	1	4	9	9





Question 5.1.1 UNION-FIND: WEIGHTED QUICK-UNION

La hauteur d'un arbre avec k noeuds en weighted quick-union est au plus $\log(k)$. Par induction.

La hauteur d'un arbre de 1 noeud est 0, donc c'est bon pour $k = 1$ ($\log(1) = 0$).

Par induction, considérons que c'est vrai pour tout $i < k$. Prenons deux arbres avec un nombre de noeud i et j tel que $i \leq j$ et $i + j = k$.

On merge donc i sur j. La hauteur des noeuds de i augmente de 1. Or

$$1 + \log(i) = \log(2) + \log(i) = \log(2i) \leq \log(i + j) = \log(k)$$

Question 5.1.1 UNION-FIND: WEIGHTED QUICK-UNION

	Find	Union
Quick-find	1	O(n)
Quick-union	O(n) Hauteur de l'arbre	O(n) Hauteur de l'arbre
Weighted quick-union	O(log(n))	O(log(n))
Better?	?	?

Question 5.1.1 UNION-FIND: PATH-COMPRESSION + WEIGHTED

SANS COMPRESSION

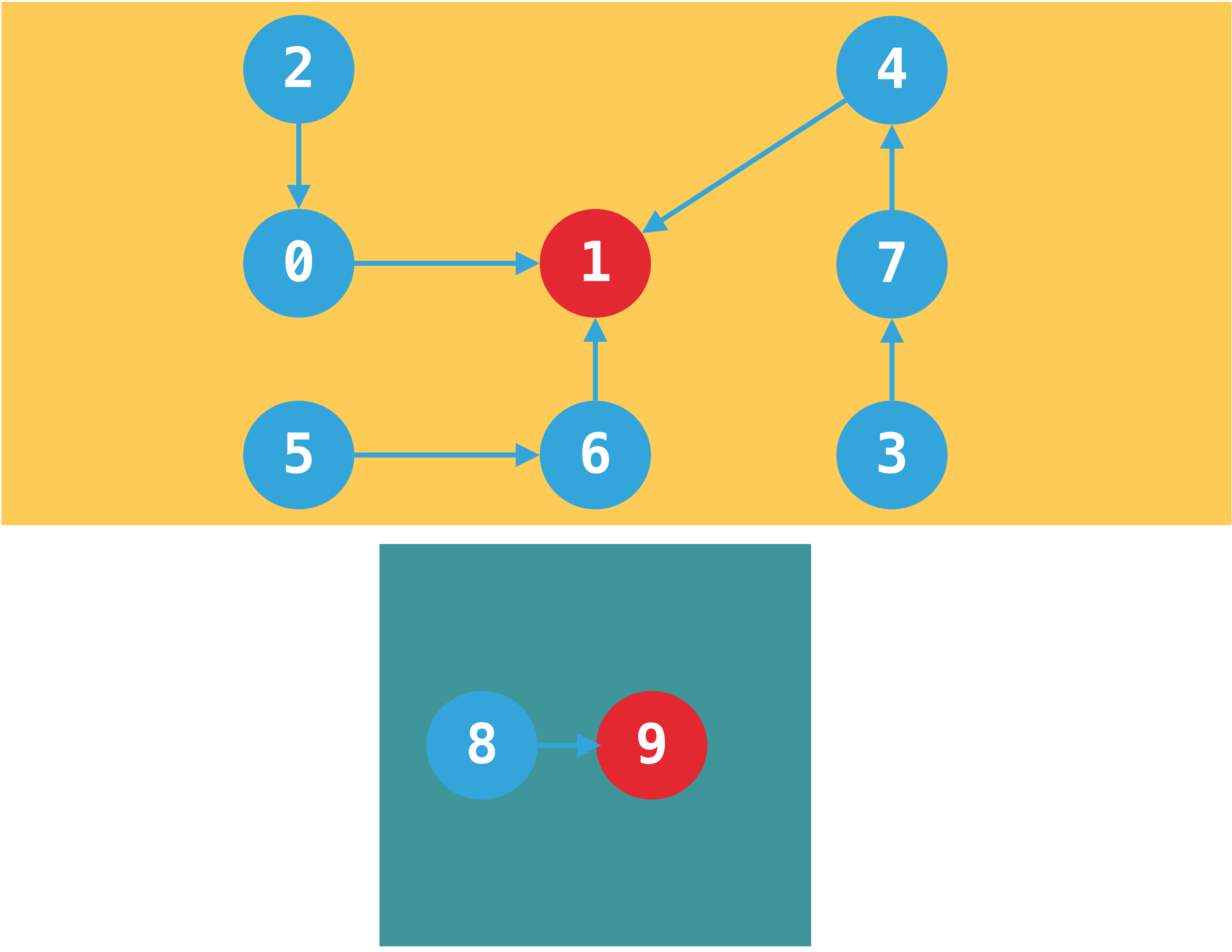
```
int find(int a) { //O(log n)
    if(tab[a] != a)
        return find(tab[a]);
    return a;
}
```

AVEC COMPRESSION

```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

Question 5.1.1 UNION-FIND: PATH COMPRESSION

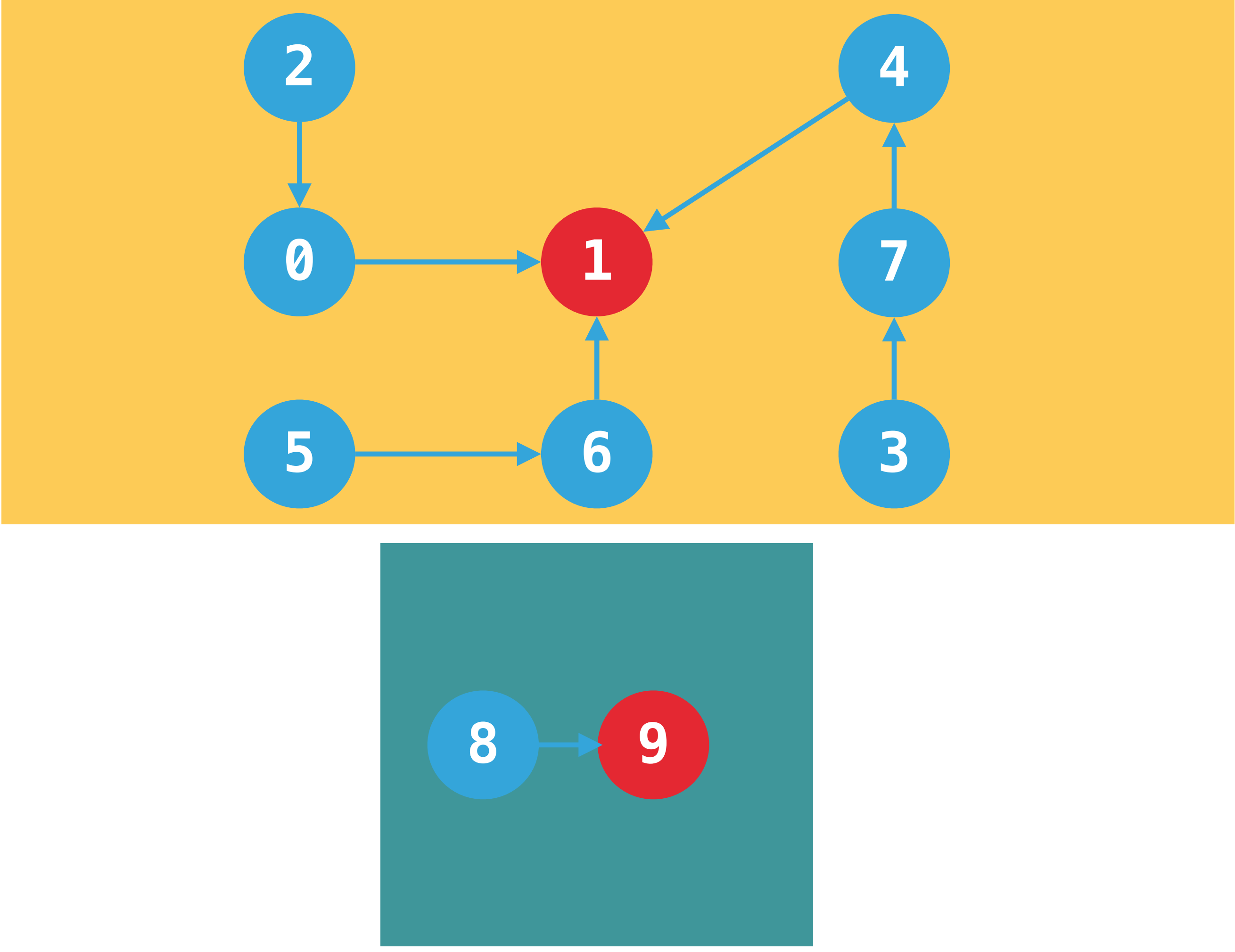
Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	0	7	1	6	1	4	9	9




```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

Question 5.1.1 UNION-FIND: PATH COMPRESSION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	0	7	1	6	1	4	9	9

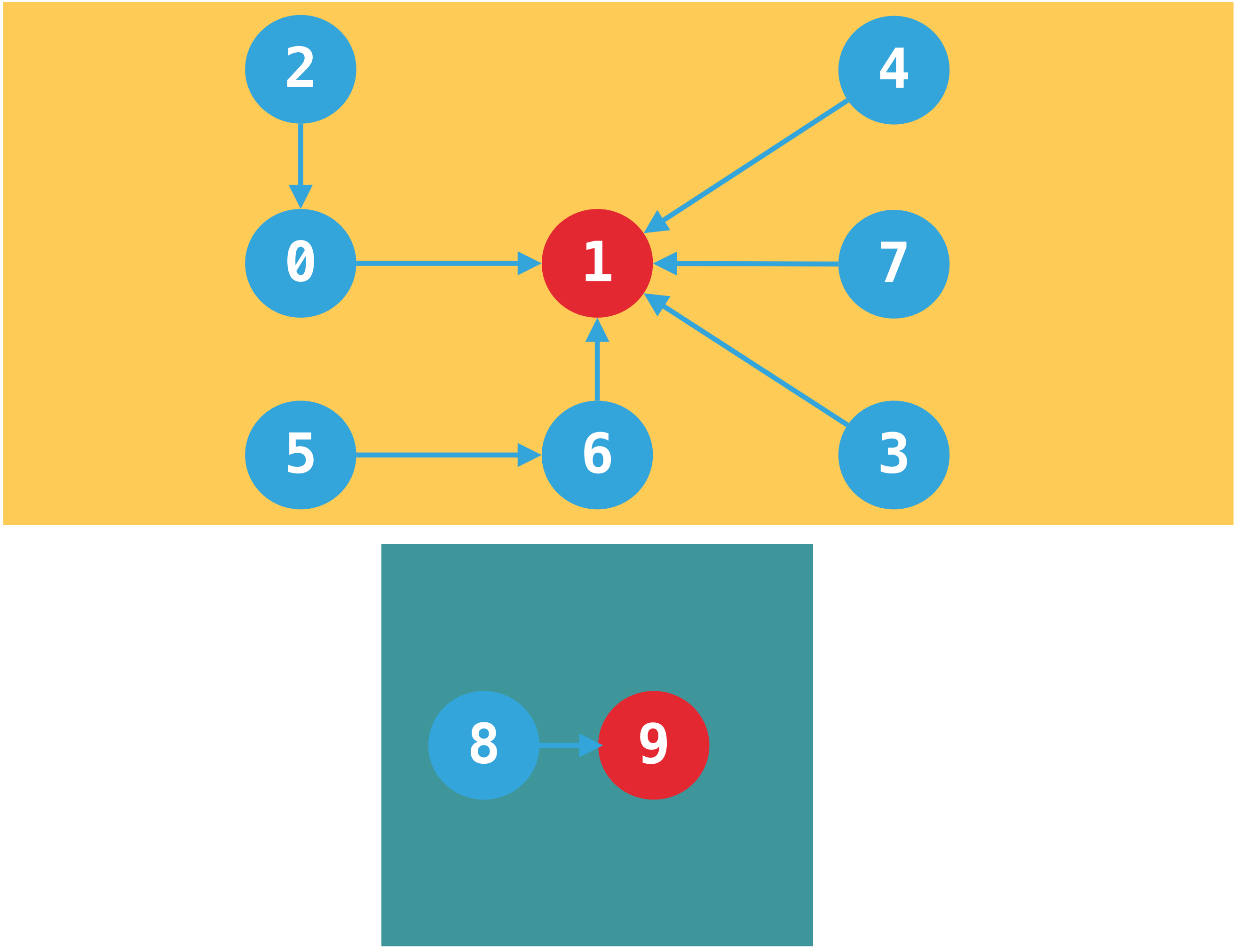



```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

FIND(3)

Question 5.1.1 UNION-FIND: PATH COMPRESSION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	0	1	1	6	1	1	9	9

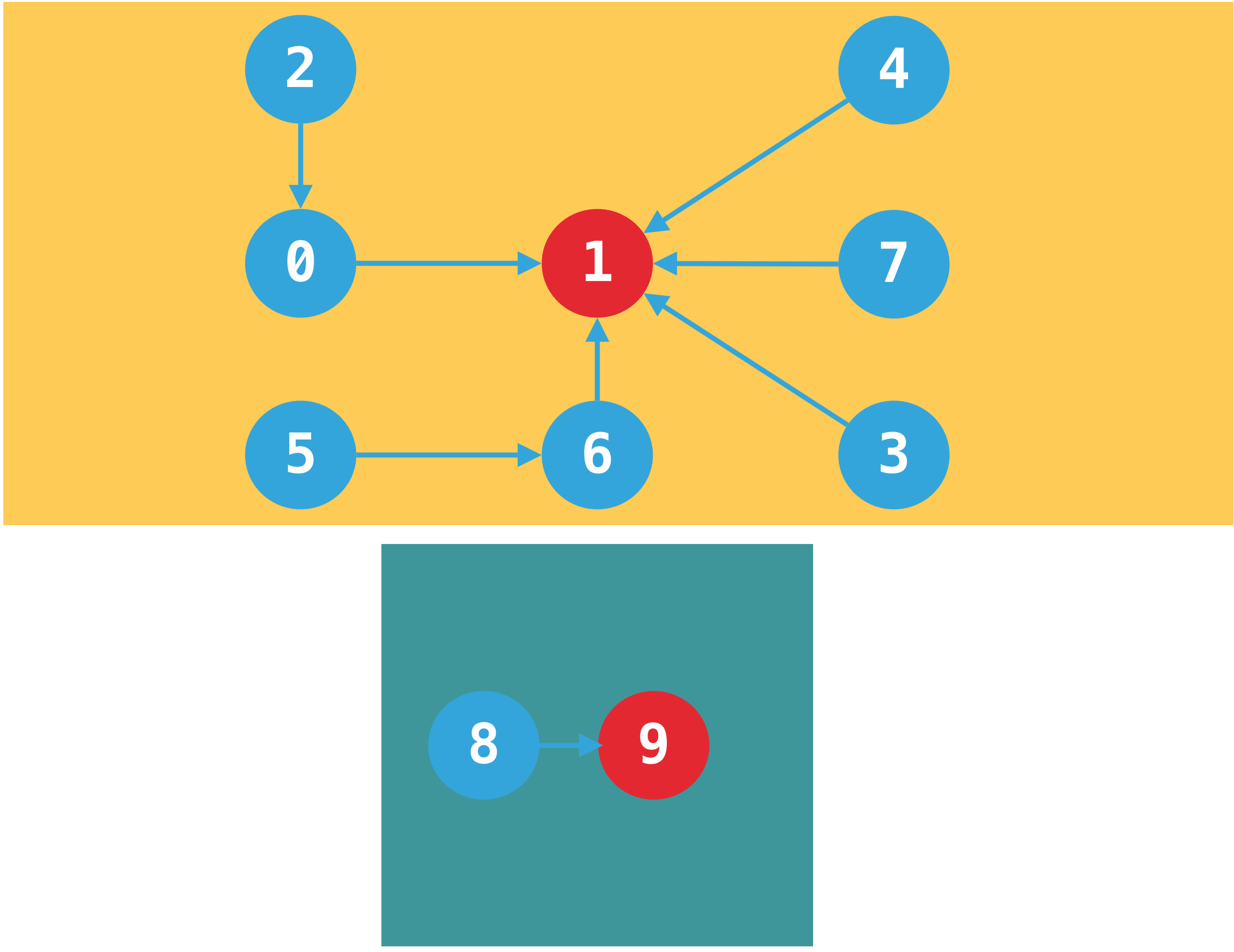



```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

FIND(3)

Question 5.1.1 UNION-FIND: PATH COMPRESSION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	0	1	1	6	1	1	9	9



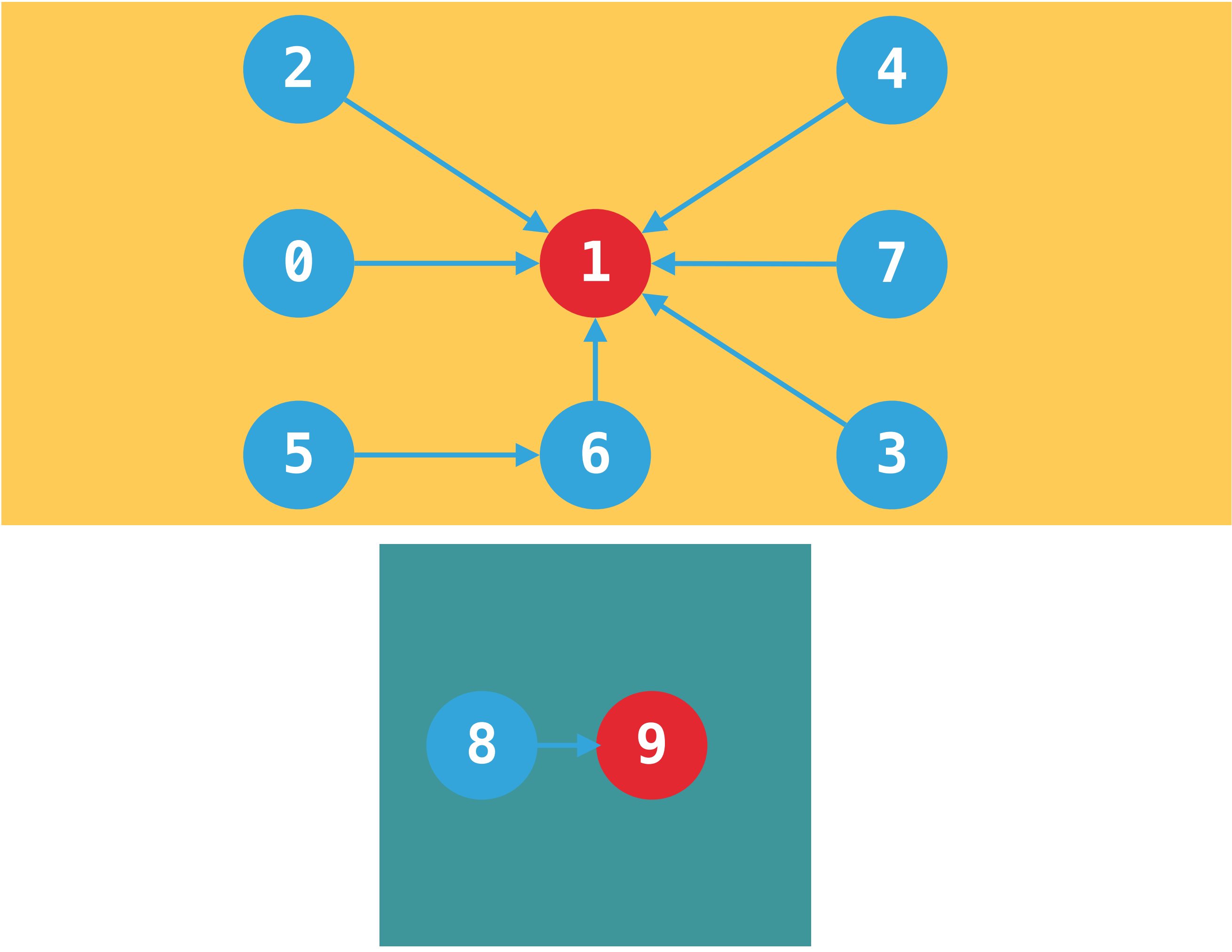

```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

FIND(3)

FIND(2)

Question 5.1.1 UNION-FIND: PATH COMPRESSION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	1	1	1	6	1	1	9	9



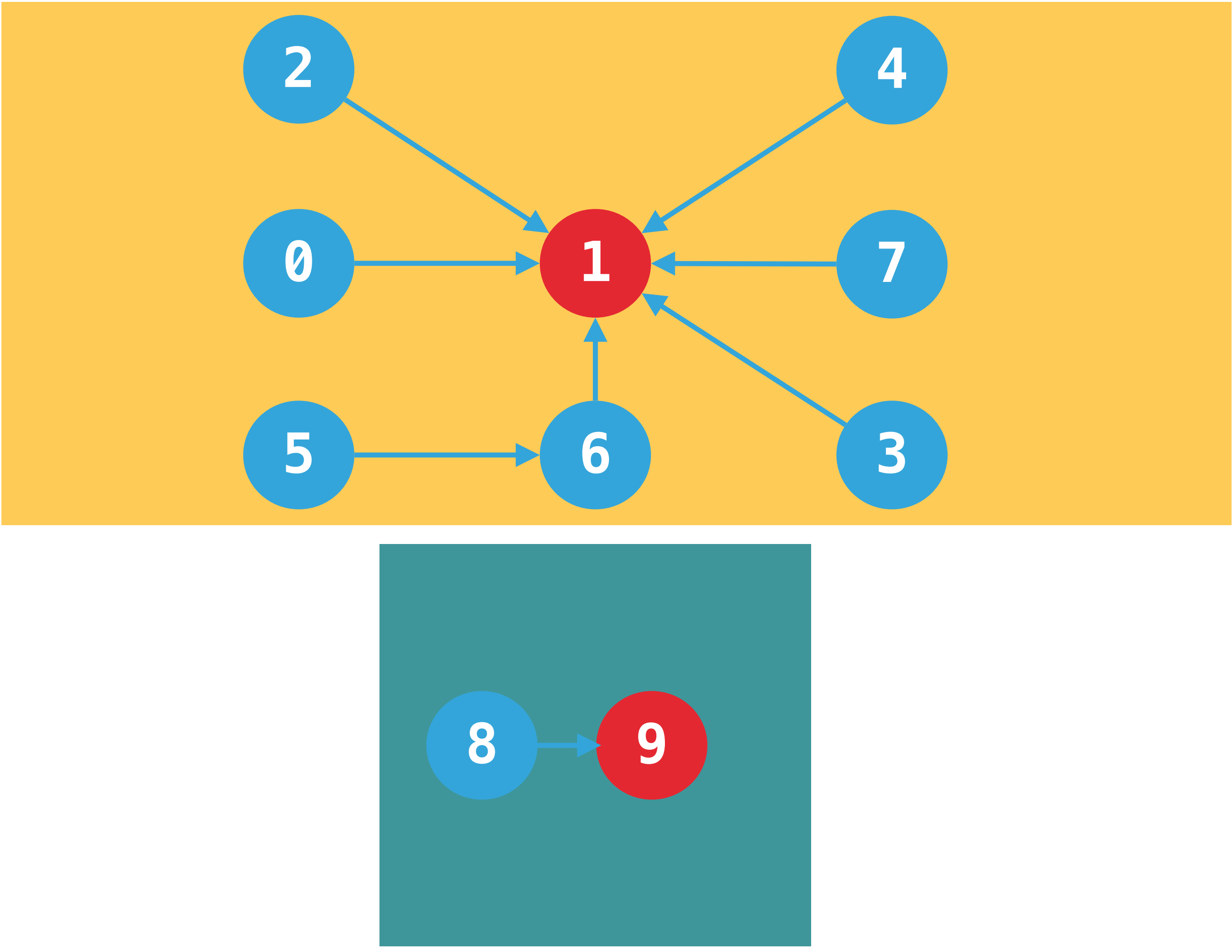

```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

FIND(3)

FIND(2)

Question 5.1.1 UNION-FIND: PATH COMPRESSION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	1	1	1	6	1	1	9	9




```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

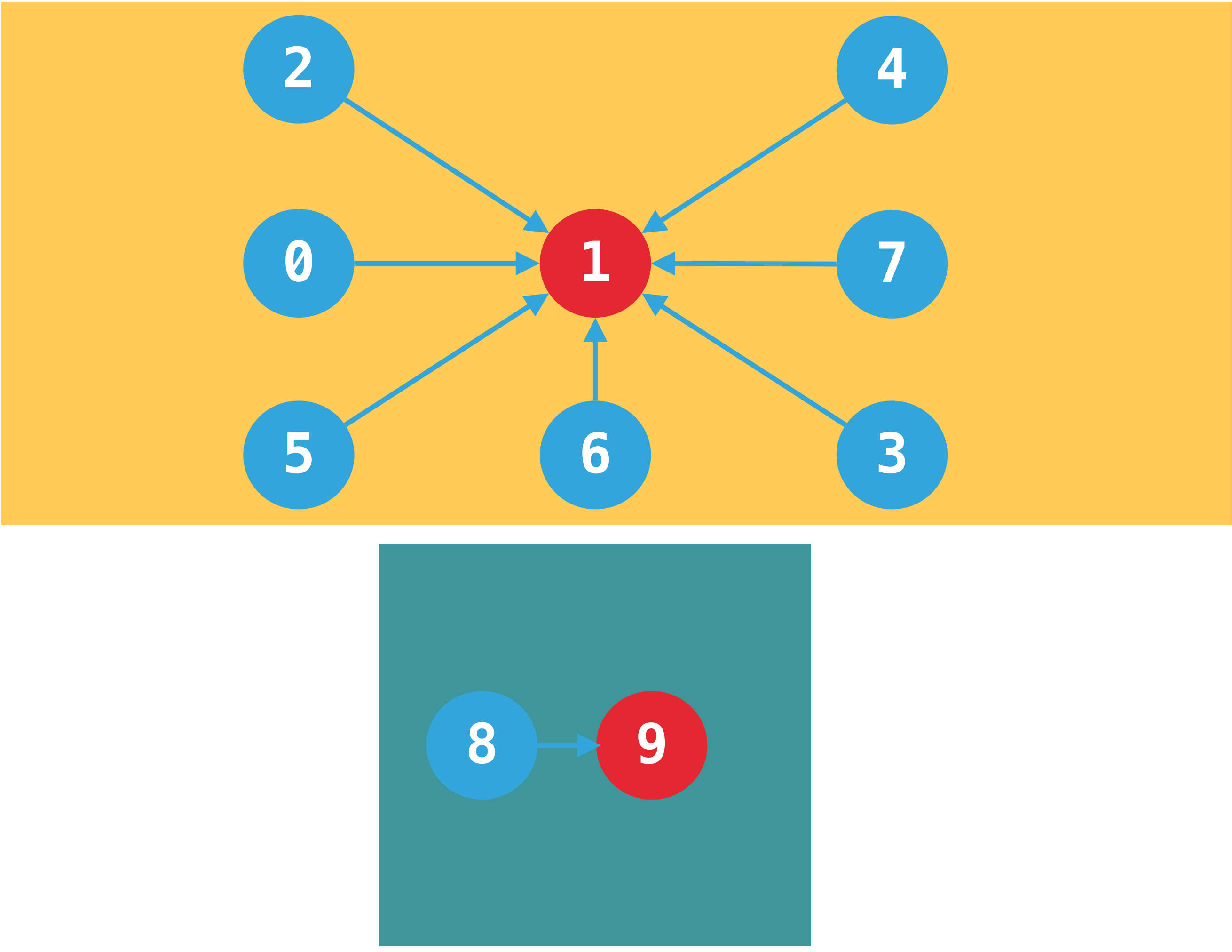
FIND(3)

FIND(2)

FIND(5)

Question 5.1.1 UNION-FIND: PATH COMPRESSION

Val	0	1	2	3	4	5	6	7	8	9
Rep	1	1	1	1	1	1	1	1	9	9




```
int find(int a) { //O(log n) ?
    if(tab[a] != a)
        tab[a] = find(tab[a]);
    return tab[a];
}
```

FIND(3)

FIND(2)

FIND(5)

Question 5.1.1 UNION-FIND: WEIGHTED QUICK-UNION

	Find	Union
Quick-find	1	O(n)
Quick-union	O(n) Hauteur de l’arbre	O(n) Hauteur de l’arbre
Weighted quick-union	O(log(n))	O(log(n))
Weighted quick-union +	< O(log(n))	< O(log(n))

Question 5.1.1 UNION-FIND: MEET ACKERMANN

$$A(m,n) = \begin{cases} n+1 & \textbf{si } m = 0 \\ A(m-1,1) & \textbf{si } m > 0 \textbf{ et } n = 0 \\ A(m-1,A(m,n-1)) & \textbf{sinon} \end{cases}$$

n	**0**	**1**	**2**	**3**	**4**
A(n,n)	1	3	7	61	$2^{2^{2^{65536}}} - 3$

La complexité de weighted quick-union avec path compression se comporte comme l'inverse de la fonction d'Ackermann $A(n,n)$:

$$\alpha(v) = n \iff A(n,n) = v$$

n	**1**	**3**	**7**	**61**	2^{64}	$2^{2^{2^{65536}}} - 3$
$\alpha(n)$	0	1	2	3	< 4	4

Question 5.1.1 UNION-FIND: WEIGHTED QUICK-UNION

	Find	Union
Quick-find	1	O(n)
Quick-union	O(n) Hauteur de l'arbre	O(n) Hauteur de l'arbre
Weighted quick-union	O(log(n))	O(log(n))
Weighted quick-union + comp	~ O(4) amorti	~ O(4) amorti

Question 5.1.1 Union-FIND

AVEC QUICK FIND!

Vale	0	1	2	3	4	5	6	7	8	9
Repr	0	1	2	3	4	5	6	7	8	9

UNION(3,8)

UNION(1,7)

UNION(1,8)

UNION(9,4)

UNION(6,4)

UNION(2,0)

Question 5.1.1 Union-FIND

AVEC QUICK FIND!

Vale	0	1	2	3	4	5	6	7	8	9
Repr	0	1	2	8	4	5	6	7	8	9

UNION(3,8)
UNION(1,7)
UNION(1,8)
UNION(9,4)
UNION(6,4)
UNION(2,0)

Question 5.1.1 Union-FIND

AVEC QUICK FIND!

Vale	0	1	2	3	4	5	6	7	8	9
Repr	0	7	2	8	4	5	6	7	8	9

UNION(3,8)
UNION(1,7)
UNION(1,8)
UNION(9,4)
UNION(6,4)
UNION(2,0)

Question 5.1.1 Union-FIND

AVEC QUICK FIND!

Vale	0	1	2	3	4	5	6	7	8	9
Repr	0	8	2	8	4	5	6	8	8	9

UNION(3,8)

UNION(1,7)

UNION(1,8)

UNION(9,4)

UNION(6,4)

UNION(2,0)

Question 5.1.1 Union-FIND

AVEC QUICK FIND!

Vale	0	1	2	3	4	5	6	7	8	9
Repr	0	8	2	8	4	5	6	8	8	4

UNION(3,8)
UNION(1,7)
UNION(1,8)
UNION(9,4)
UNION(6,4)
UNION(2,0)

Question 5.1.1 Union-FIND

AVEC QUICK FIND!

Vale	0	1	2	3	4	5	6	7	8	9
Repr	0	8	2	8	4	5	4	8	8	4

UNION(3,8)

UNION(1,7)

UNION(1,8)

UNION(9,4)

UNION(6,4)

UNION(2,0)

Question 5.1.1 Union-FIND

AVEC QUICK FIND!

Vale	0	1	2	3	4	5	6	7	8	9
Repr	0	8	0	8	4	5	4	8	8	4

UNION(3,8)
UNION(1,7)
UNION(1,8)
UNION(9,4)
UNION(6,4)
UNION(2,0)

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	0	1	2	3	4	5	6	7	8	9
Size	1	1	1	1	1	1	1	1	1	1

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Vale	0	1	2	3	4	5	6	7	8	9
Rep	0	1	2	3	4	5	4	7	8	9
Size	1	1	1	1	2	1		1	1	1

- UNION(4,6)
- UNION(3,6)
- UNION(8,9)
- UNION(7,0)
- UNION(1,2)
- UNION(8,4)
- UNION(6,5)
- UNION(1,7)
- UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	0	1	2	4	4	5	4	7	8	9
Size	1	1	1		3	1		1	1	1

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Valeur	0	1	2	3	4	5	6	7	8	9
Rep	0	1	2	4	4	5	4	7	8	8
Size	1	1	1		3	1		1	2	

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Vale	0	1	2	3	4	5	6	7	8	9
Rep	7	1	2	4	4	5	4	7	8	8
Size		1	1		3	1		2	2	

UNION(4,6)

UNION(3,6)

UNION(8,9)

UNION(7,0)

UNION(1,2)

UNION(8,4)

UNION(6,5)

UNION(1,7)

UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	7	1	1	4	4	5	4	7	8	8
Size		2			3	1		2	2	

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	7	1	1	4	4	5	4	7	4	8
Size		2			5	1		2		

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	7	1	1	4	4	4	4	7	4	8
Size		2			6			2		

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	7	1	1	4	4	4	4	1	4	8
Size		4			6					

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d'égalité, on fait pointer le second arbre vers le premier.

Question 5.1.2 Union-FIND

AVEC WEIGHTED QUICK UNION!

Val	0	1	2	3	4	5	6	7	8	9
Rep	7	4	1	4	4	4	4	1	4	8
Siz					10					

UNION(4,6)
UNION(3,6)
UNION(8,9)
UNION(7,0)
UNION(1,2)
UNION(8,4)
UNION(6,5)
UNION(1,7)
UNION(6,0)

En cas d’égalité, on fait pointer le second arbre vers le premier.

Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	0	8	2	3	4	7	6	8	8	9
Size										

Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	0	8	2	3	4	7	6	8	8	9
Size	1		1	1	1		1		4	1

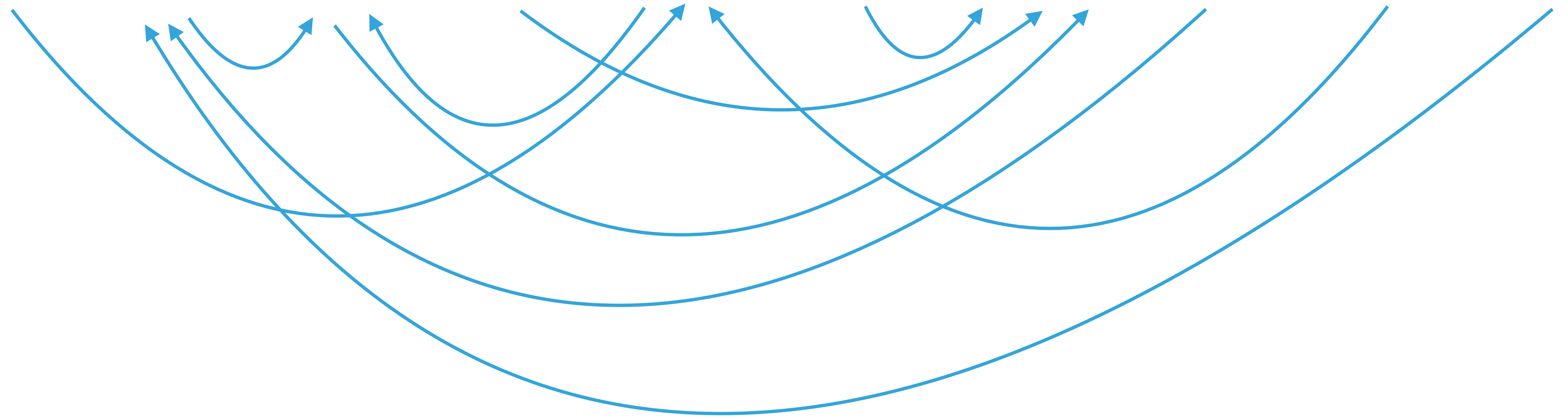
UNION(8,1)
UNION(7,5)
UNION(7,8)

Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	4	2	6	6	2	6	6	2	4	2
Size										

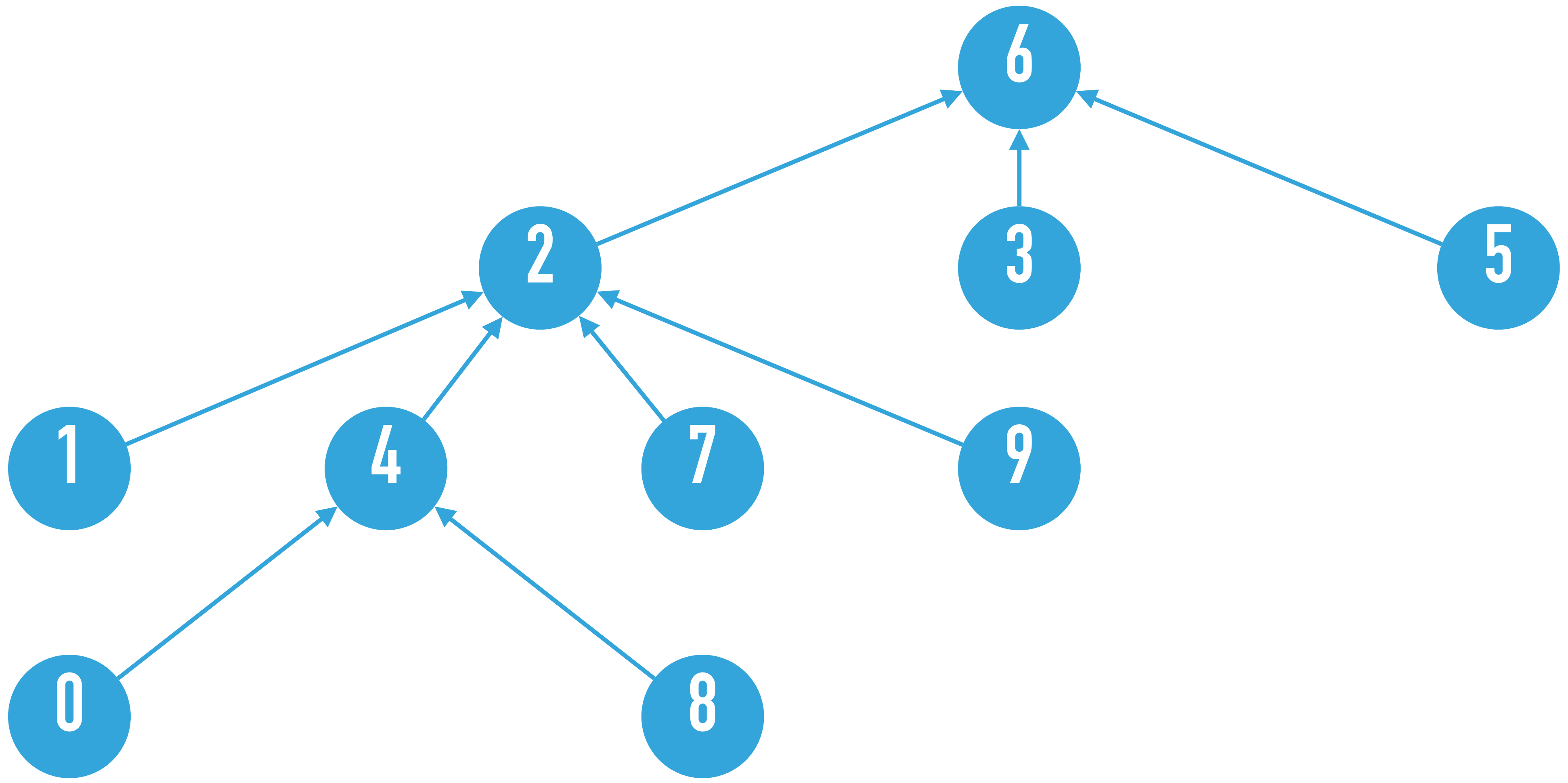
Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	4	2	6	6	2	6	6	2	4	2
Size							10			



Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	4	2	6	6	2	6	6	2	4	2
Size	1	1	7	1	3	1	10	1	1	1

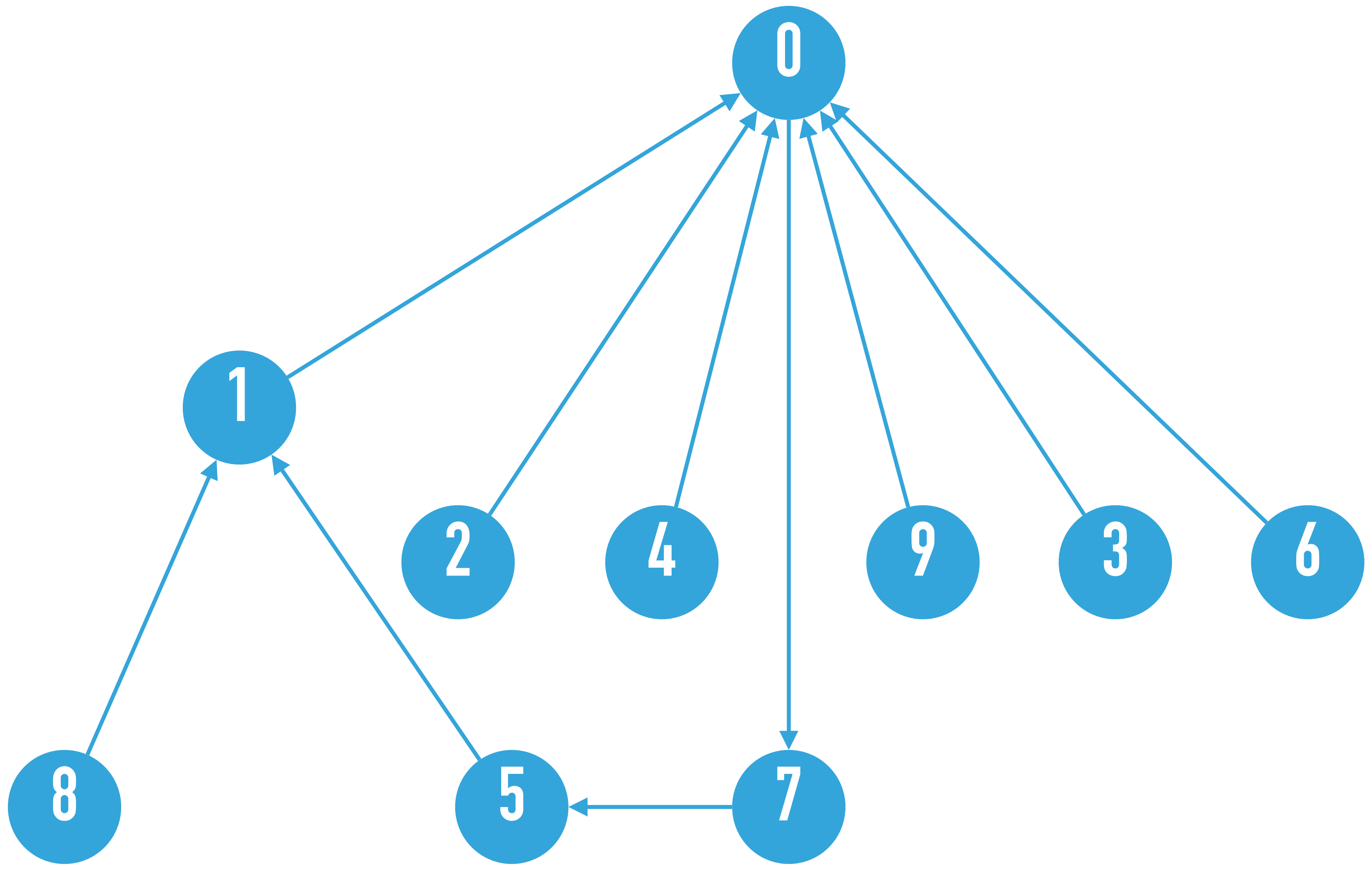


Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	7	0	0	0	0	1	0	5	1	0
Size										

Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	7	0	0	0	0	1	0	5	1	0
Size										

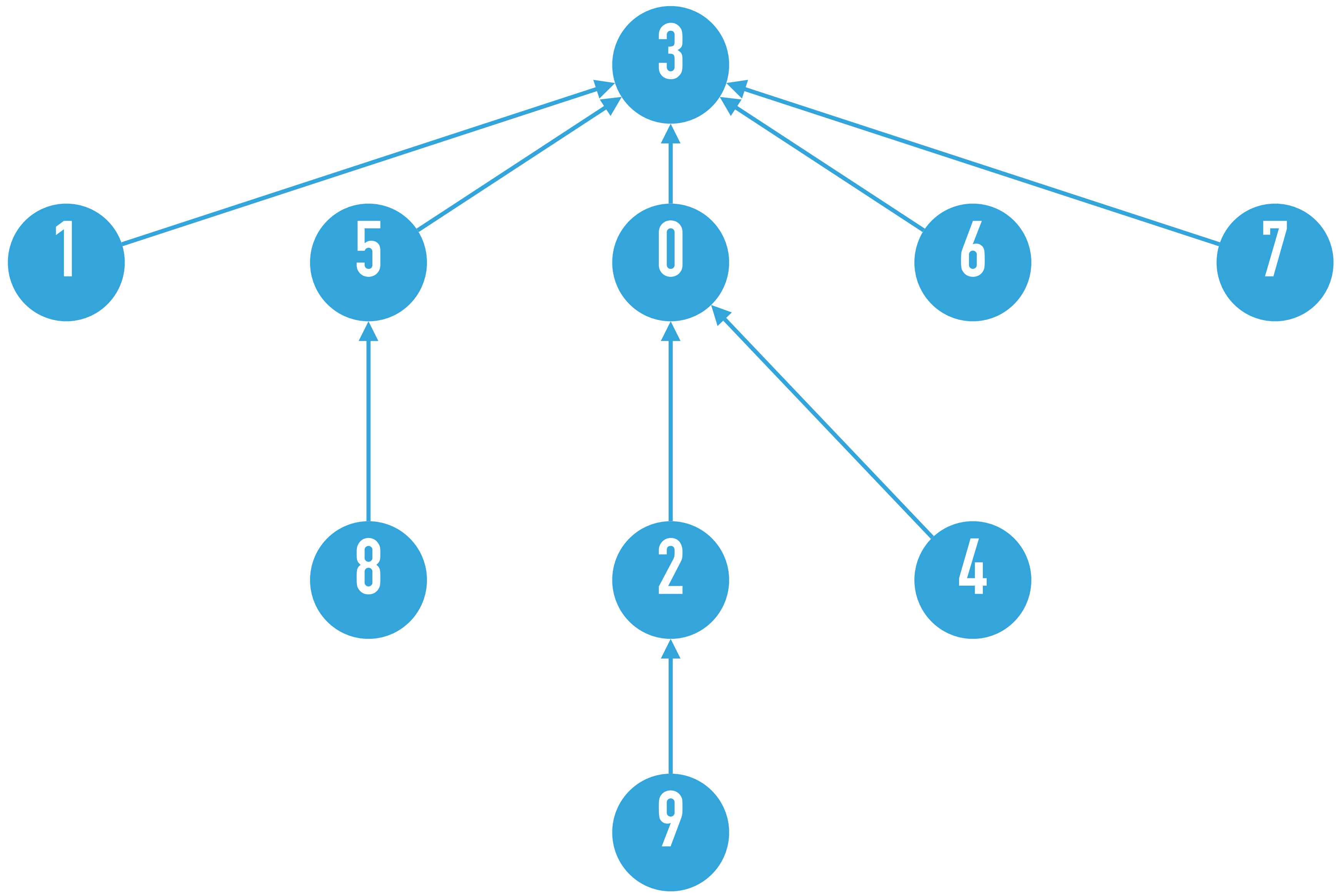


Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	3	3	0	3	0	3	3	3	5	2
Size										

Question 5.1.3 weighted-quick-union: possible?

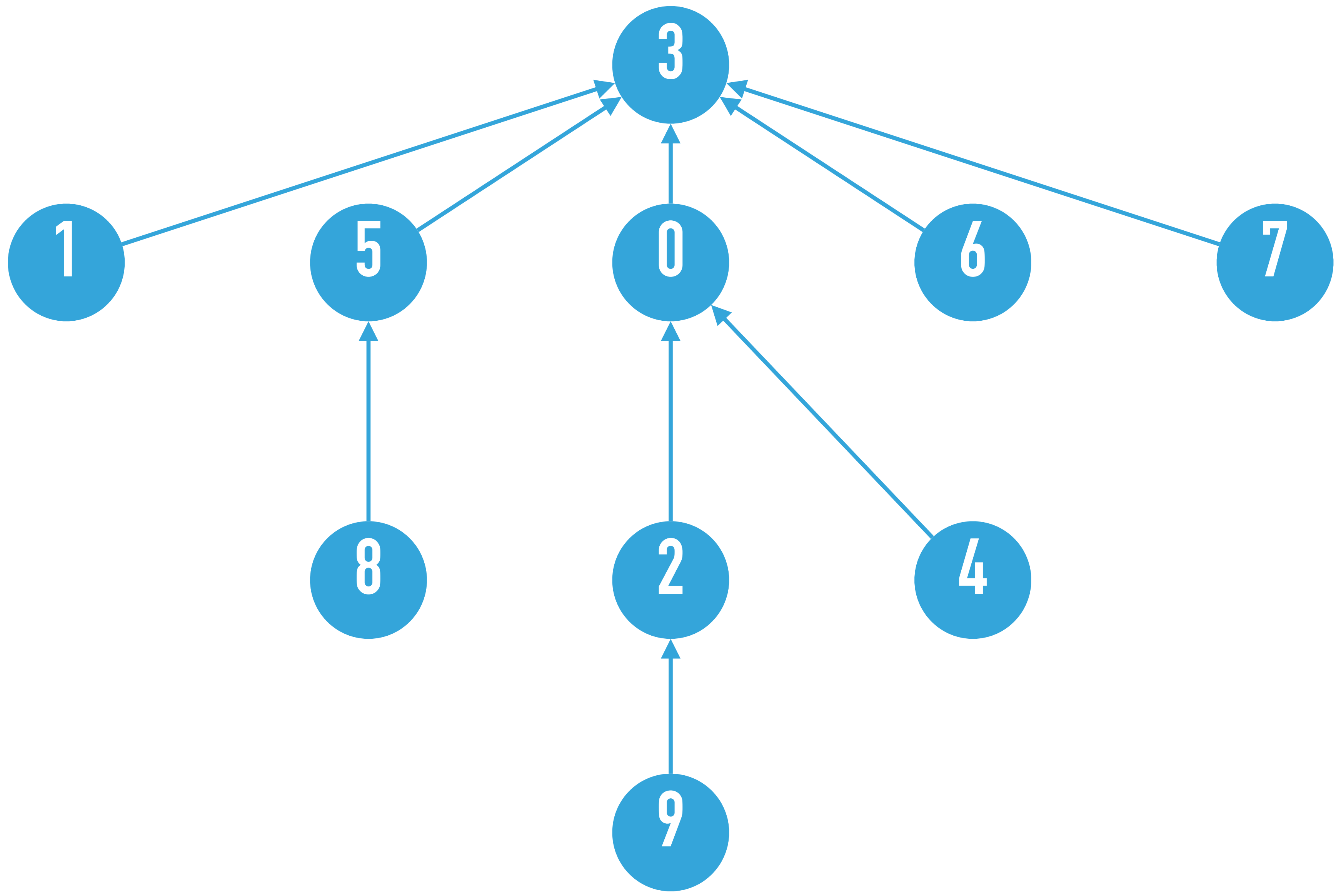
Valeur	0	1	2	3	4	5	6	7	8	9
Repr	3	3	0	3	0	3	3	3	5	2
Size	4	1	2	10	1	2	1	1	1	1



Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	3	3	0	3	0	3	3	3	5	2
Size	4	1	2	10	1	2	1	1	1	1

UNION(3,1)
UNION(3, 6)
UNION(3, 7)
UNION(5, 8)
UNION(2,9)
UNION(0, 4)
UNION(0, 2)
UNION(3, 5)
UNION(3, 0)

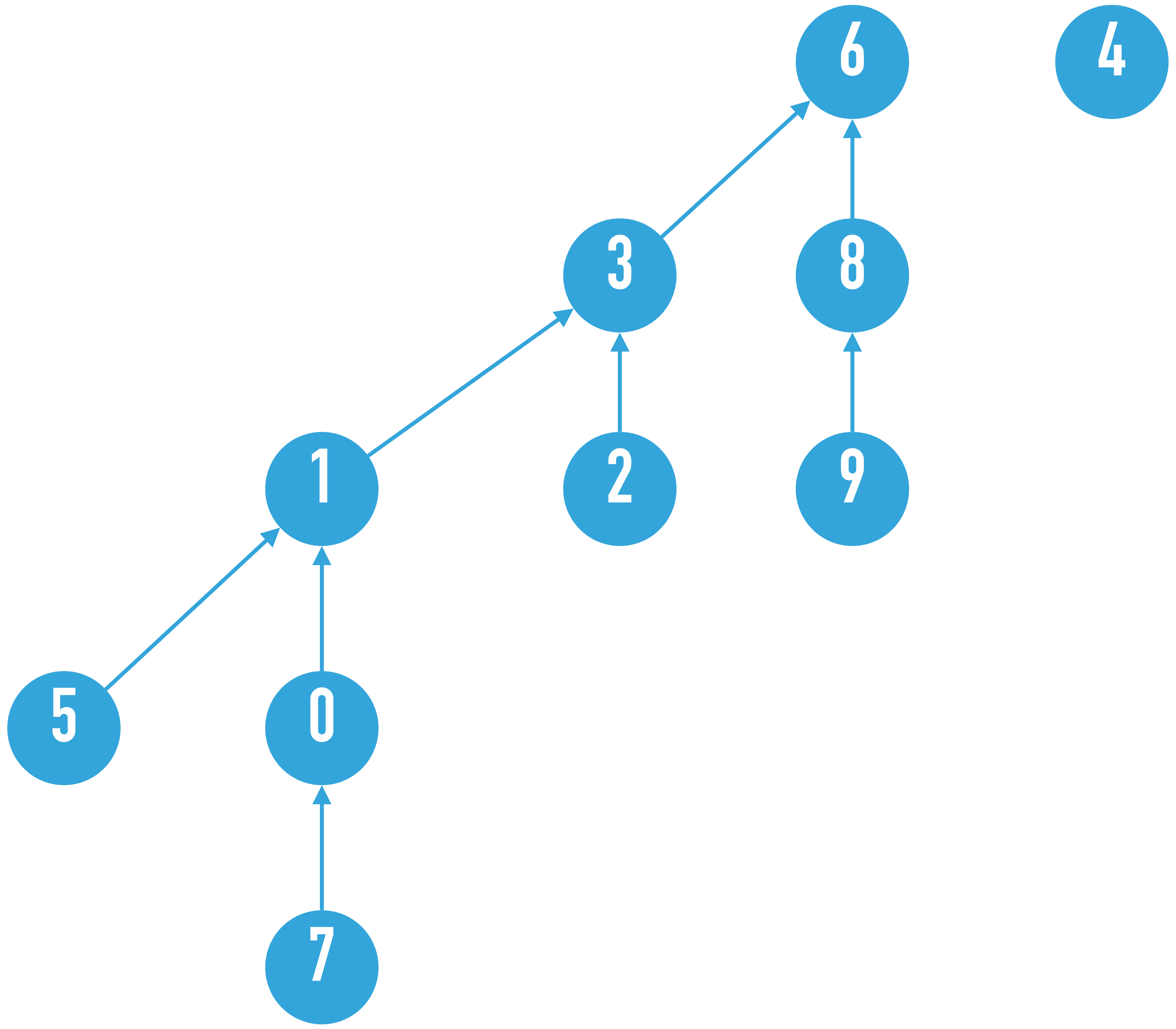


Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	1	3	3	6	4	1	6	0	6	8
Size										

Question 5.1.3 weighted-quick-union: possible?

Valeur	0	1	2	3	4	5	6	7	8	9
Repr	1	3	3	6	4	1	6	0	6	8
Size	2	4	1	6	1	1	9	1	2	1



Question 5.1.4 PRIORITY QUEUES

Une priority queue est un ADT qui offre (principalement) les opérations suivantes:

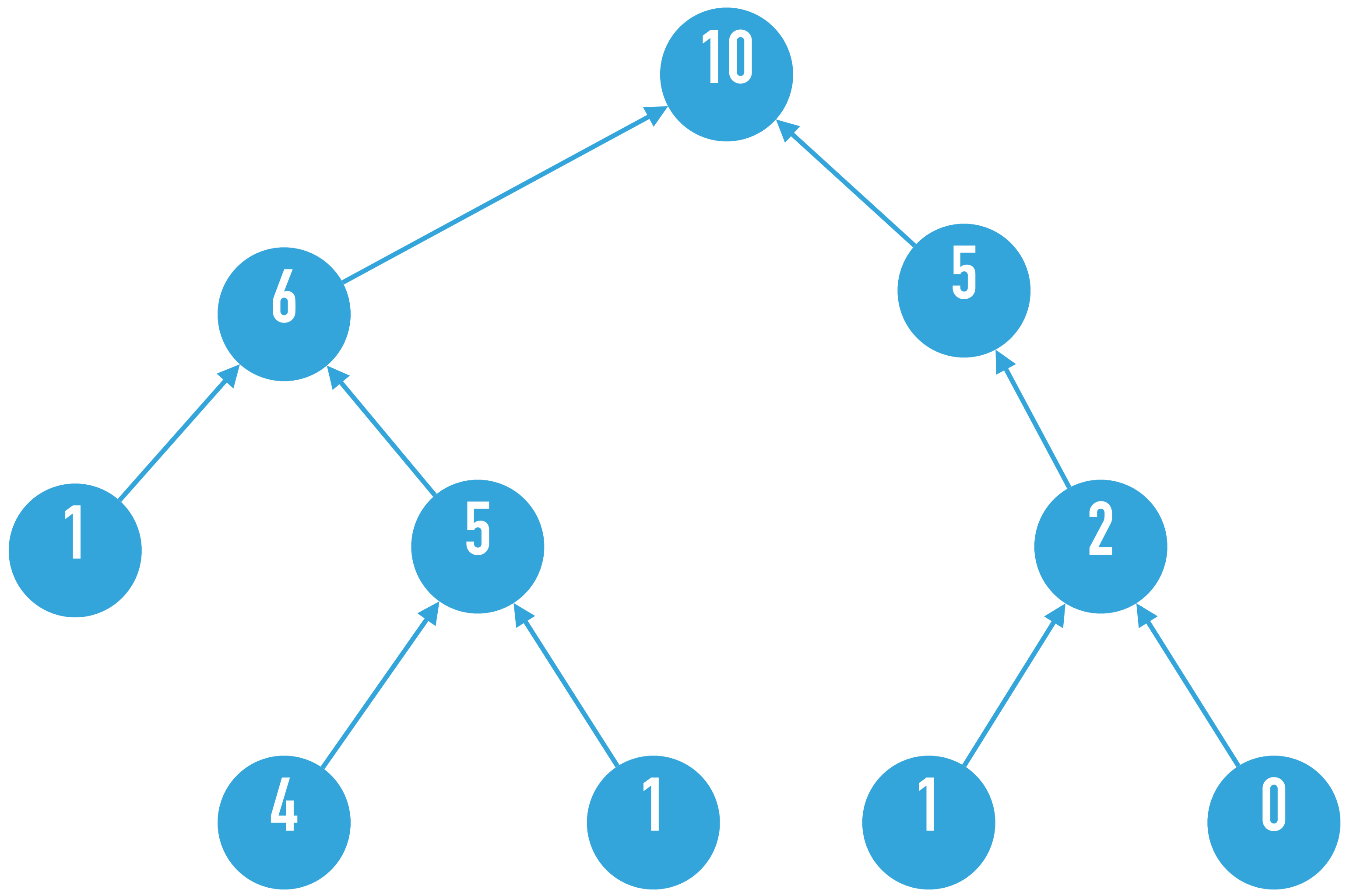
- Ajouter une valeur i
- Retirer la plus grande valeur de la priority queue

On utilise généralement des complete binary heaps pour implémenter ces deux opérations rapidement.

Question 5.1.4 HEAPS

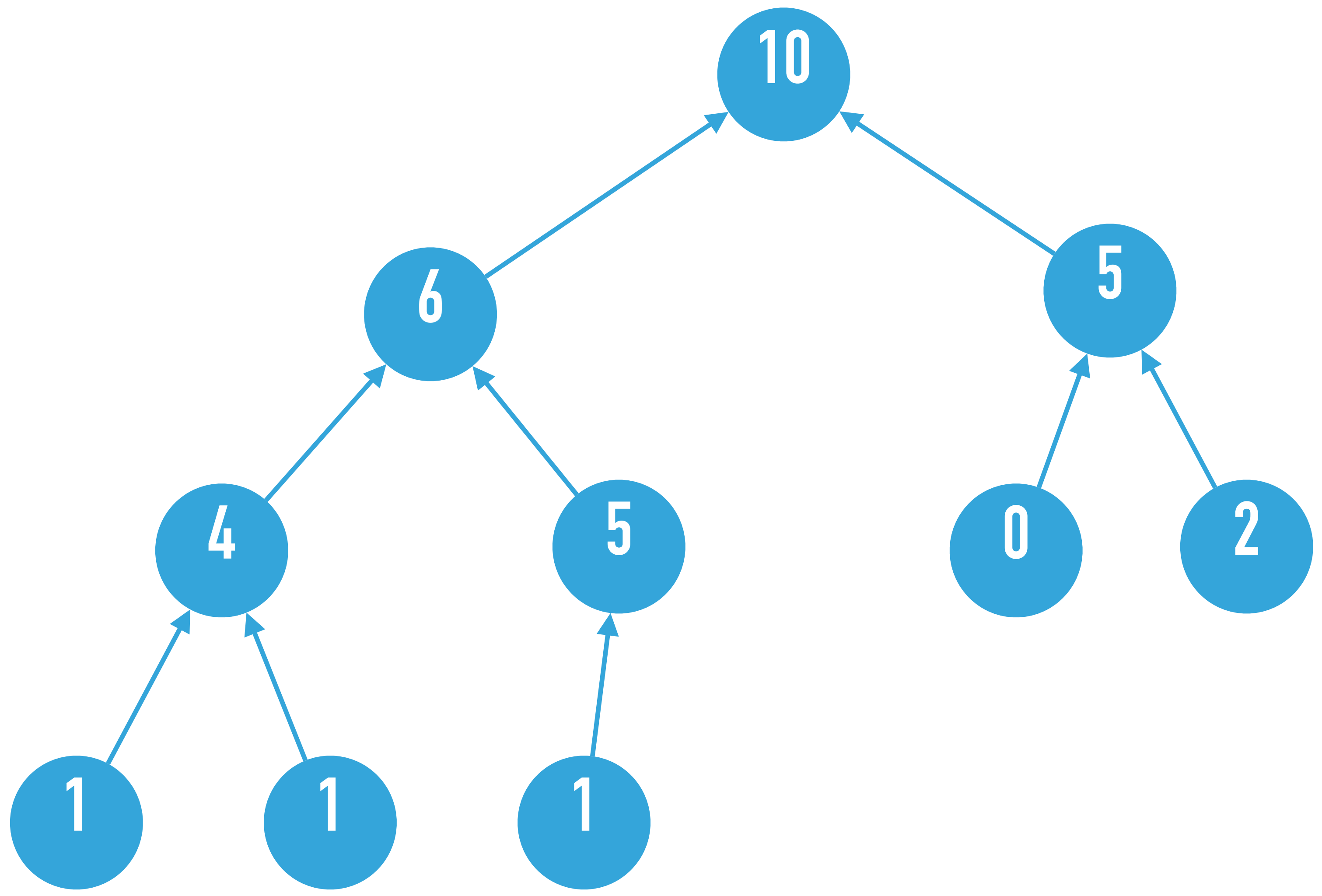
Une heap est une implémentation d'une priority queue.

Une heap binaire est un arbre binaire telle que la clé de n'importe quel noeud est supérieure à celles de ces enfants



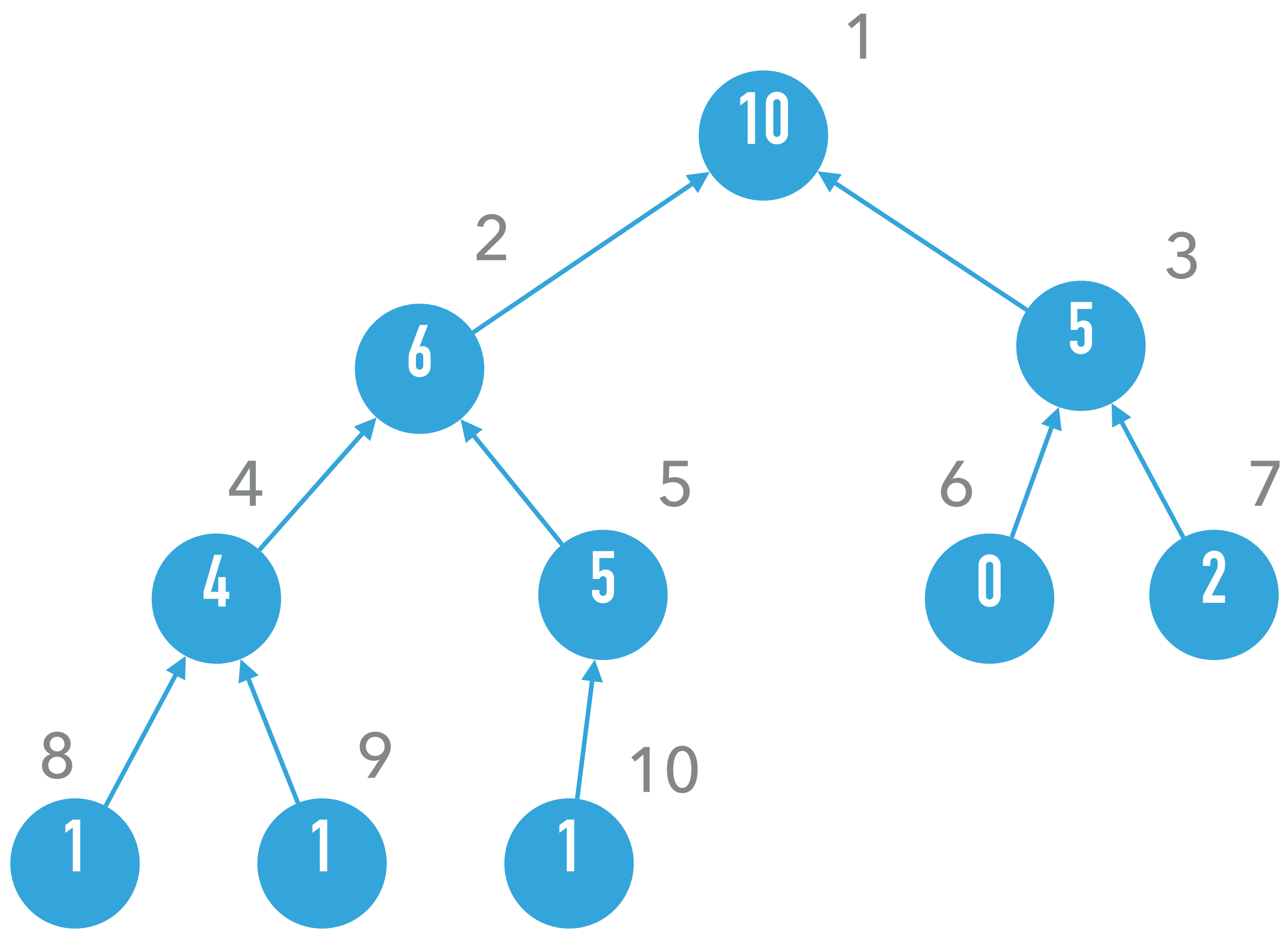
Question 5.1.4 HEAPS

Nous allons utiliser des heaps complets: l'arbre est rempli et la dernière couche situe ces noeuds en bas à gauche prioritairement.



Question 5.1.4 HEAPS

Les arbres binaires complets peuvent être représentés aisément par un tableau:



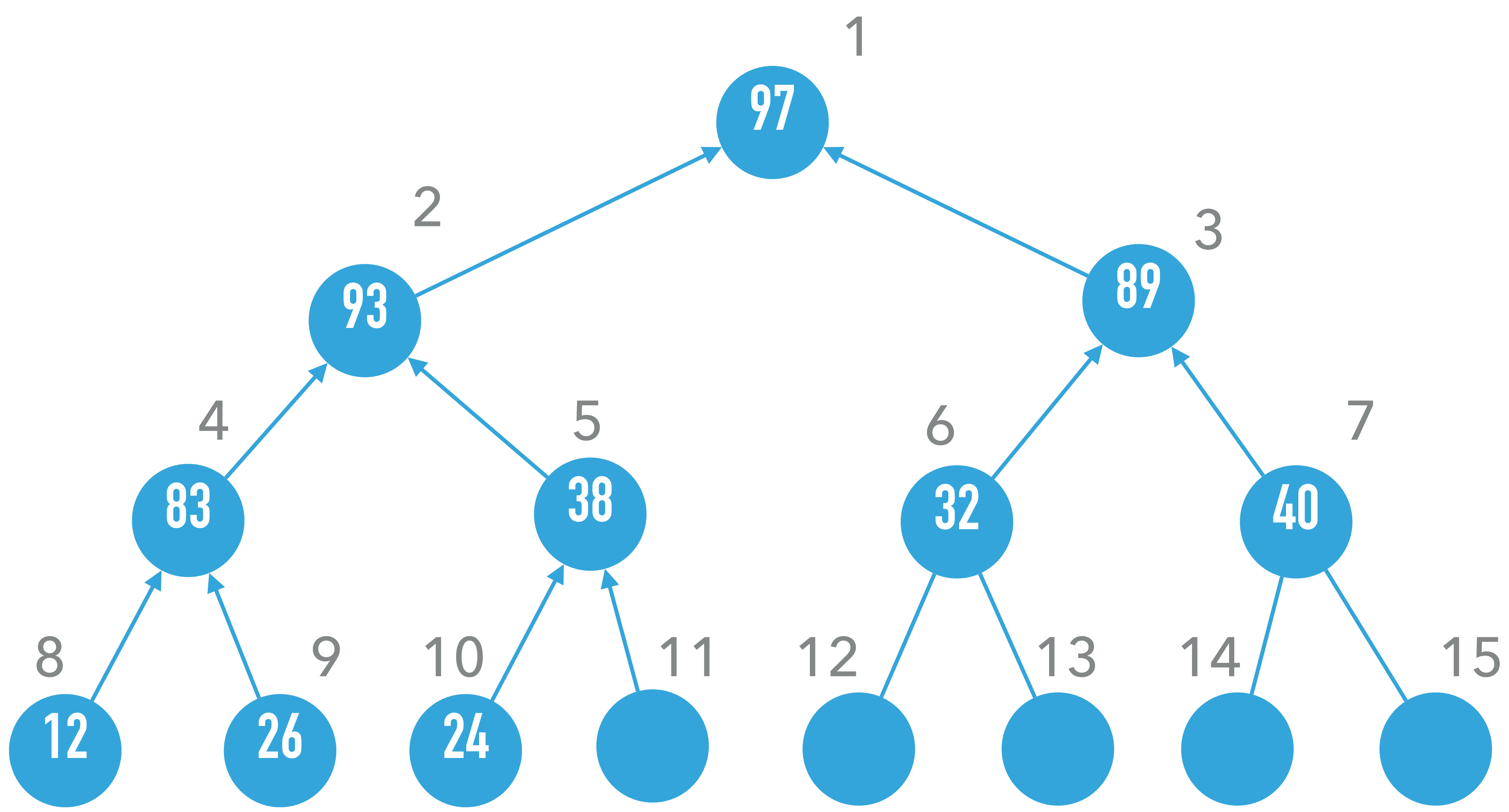
Idx	1	2	3	4	5	6	7	8	9	10
V	10	6	5	4	5	0	2	1	1	1

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	38	32	40	12	26	24					



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

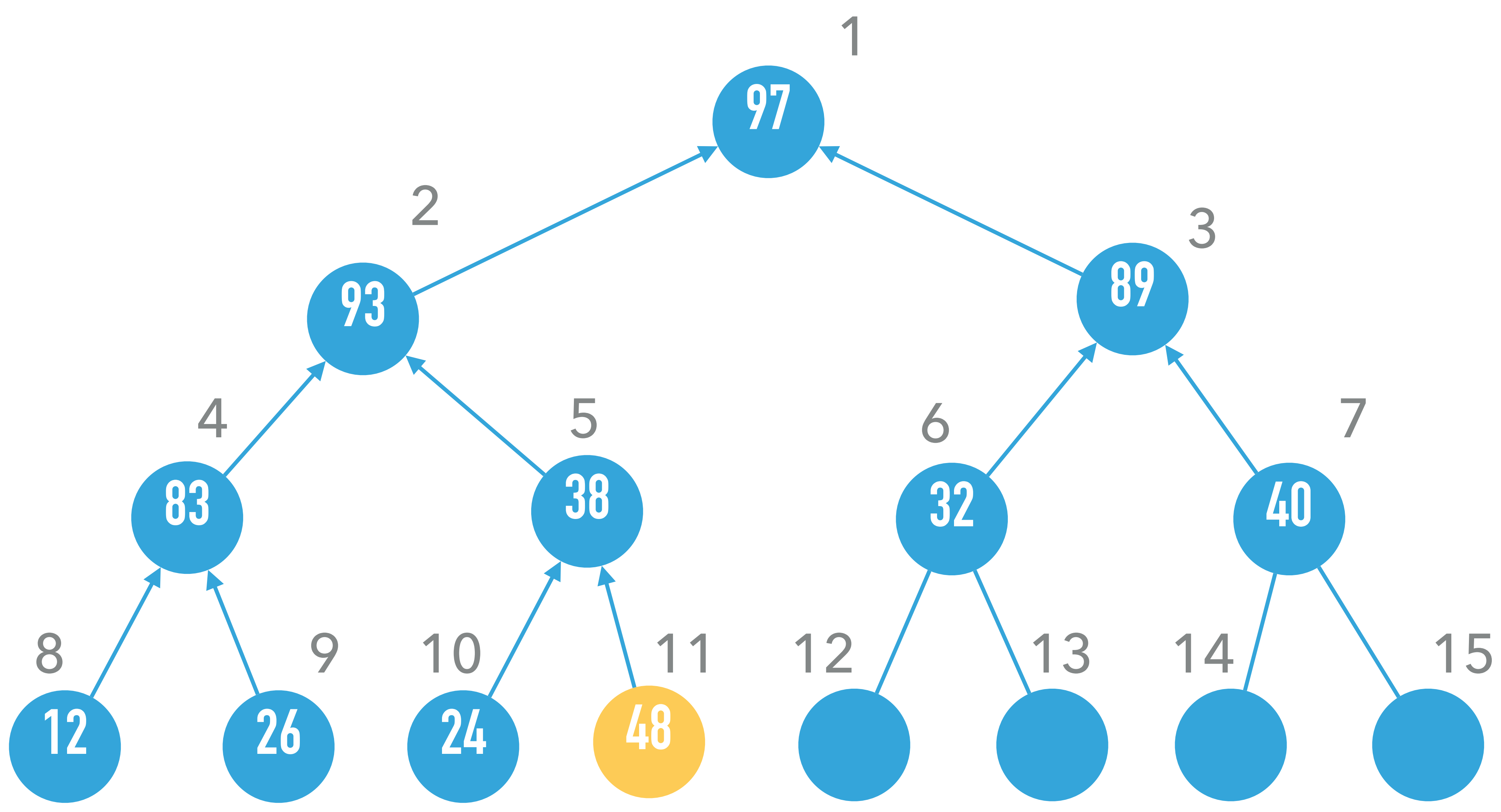
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	38	32	40	12	26	24	48				



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

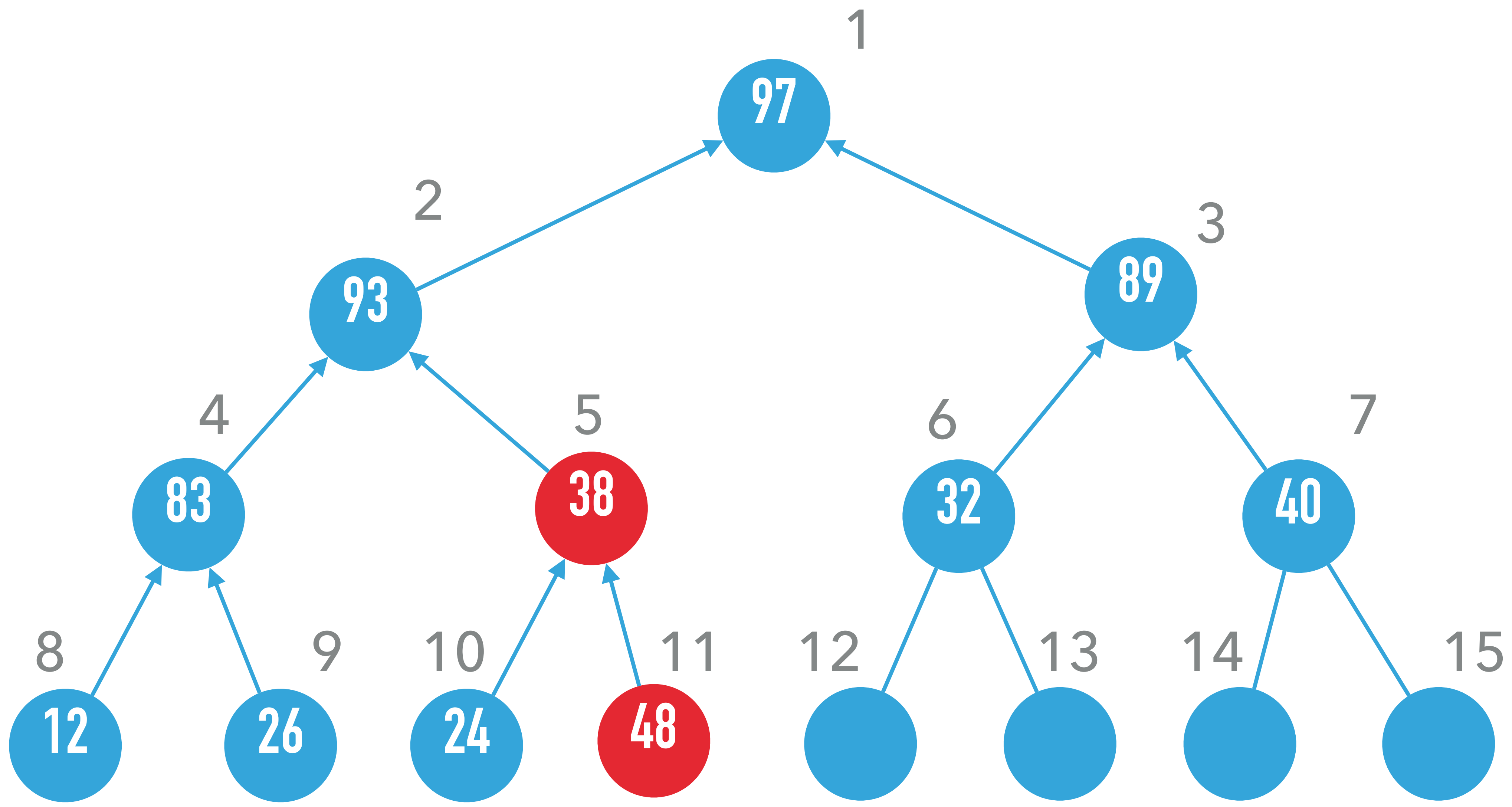
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	38	32	40	12	26	24	48				



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

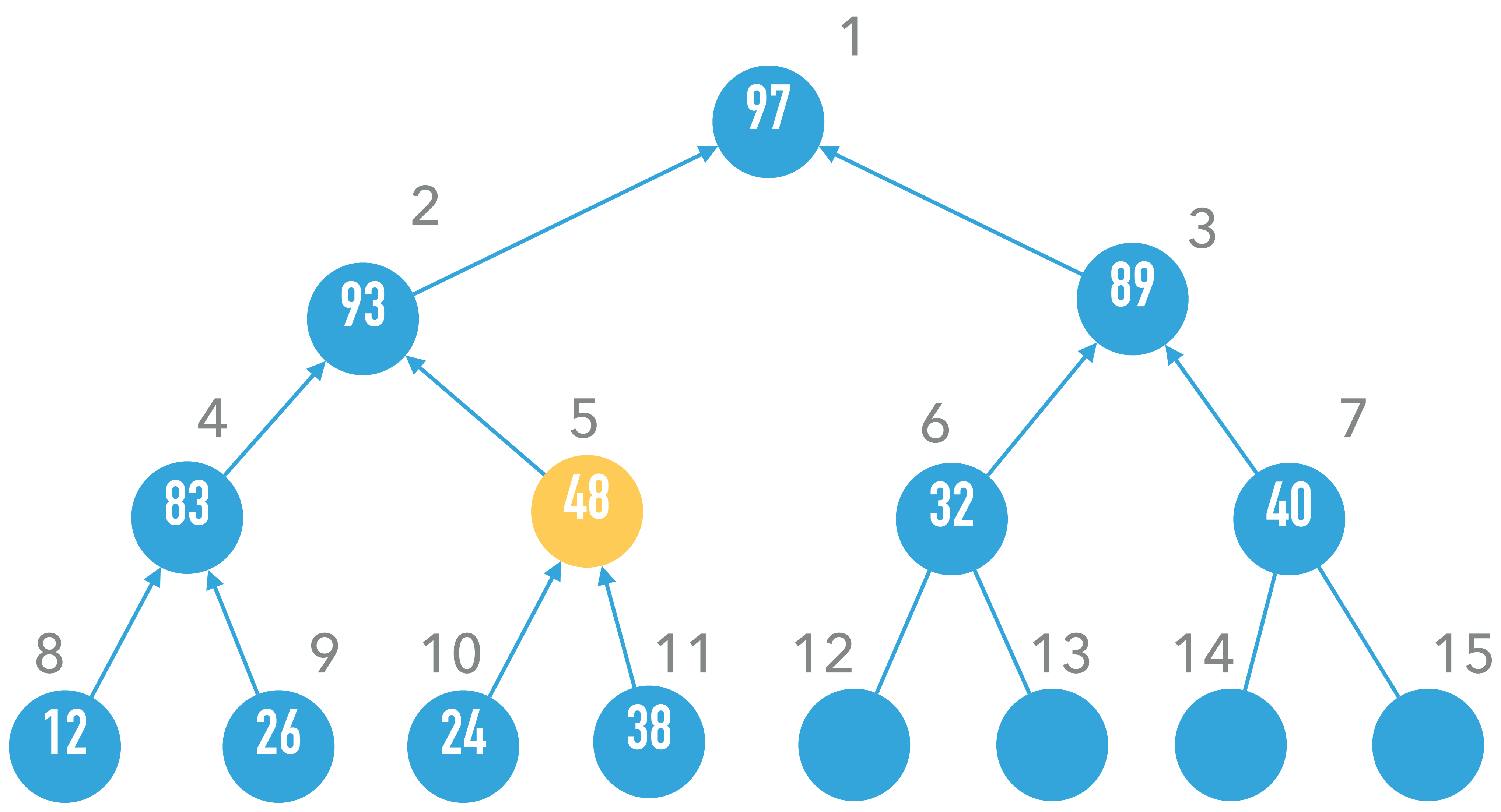
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38				



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

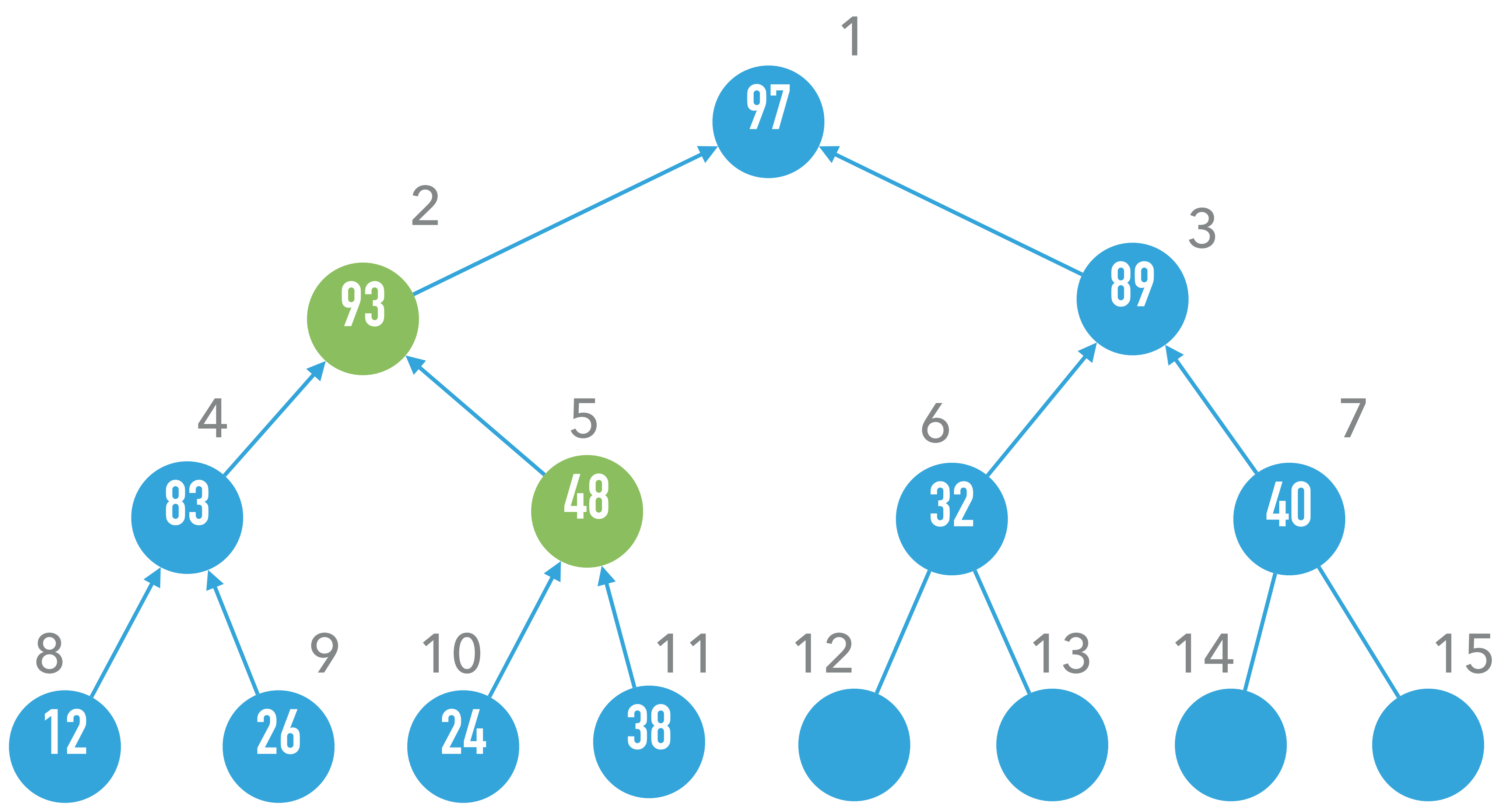
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38				



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

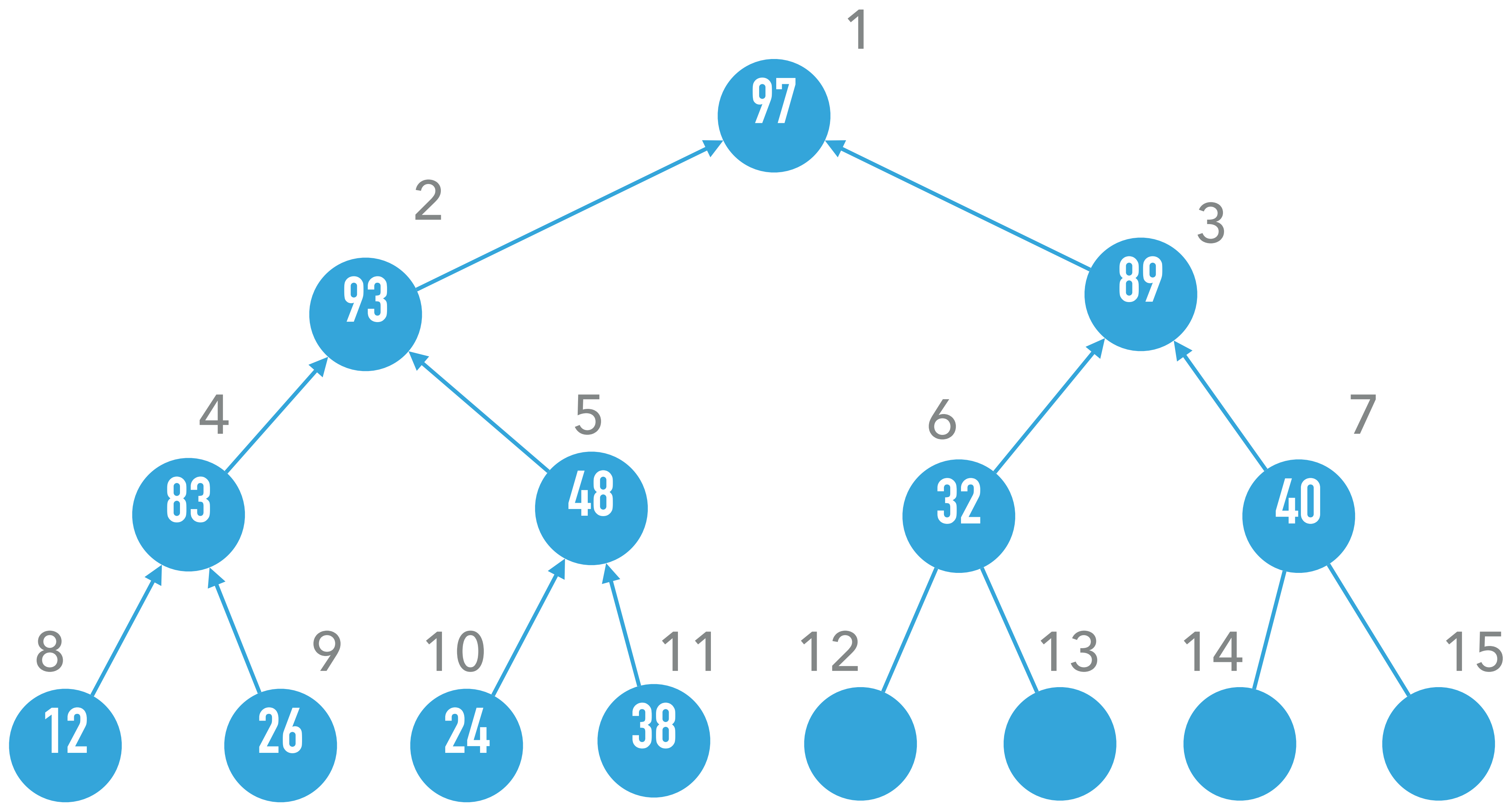
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38				



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

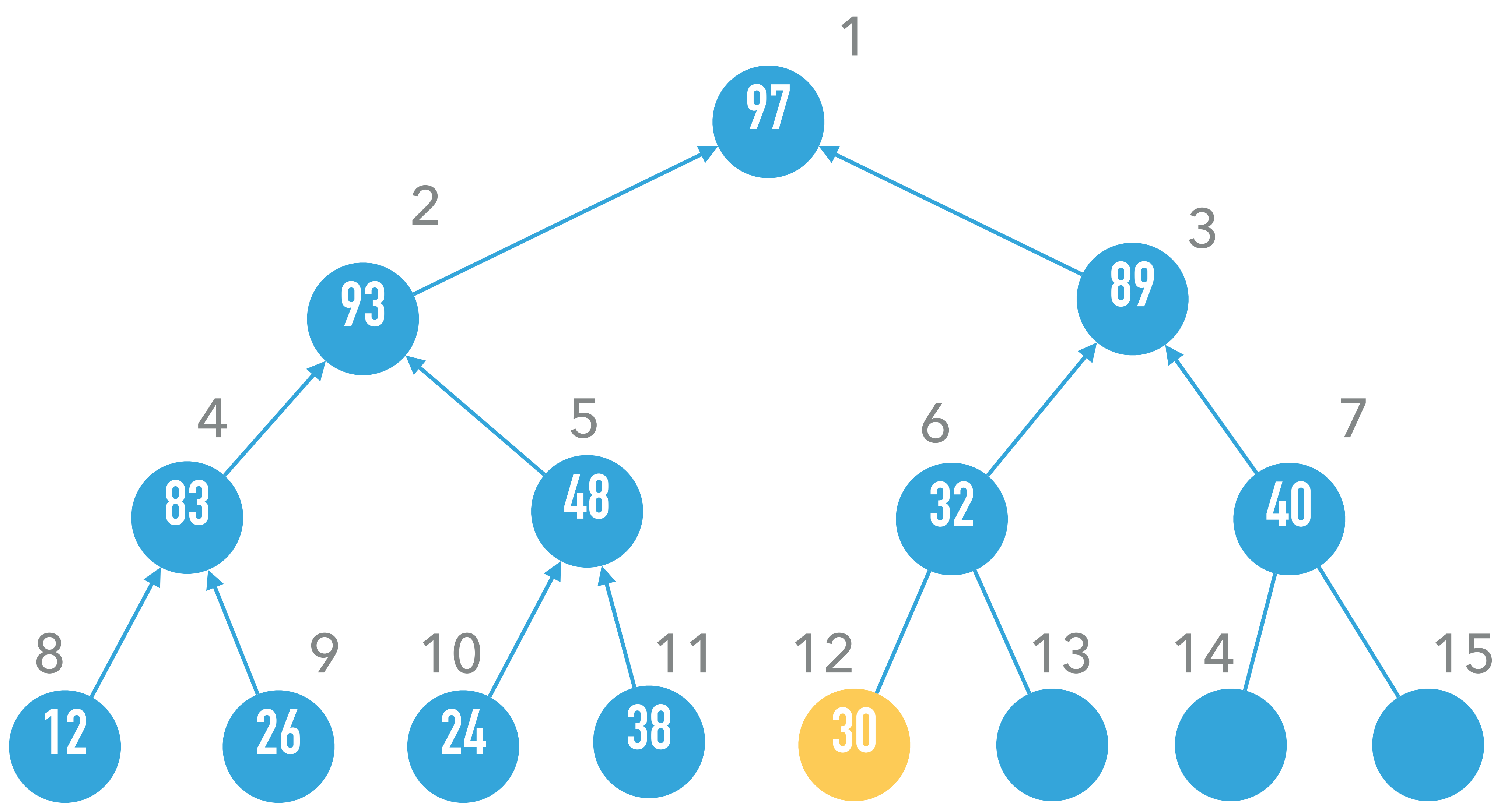
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38	30			



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

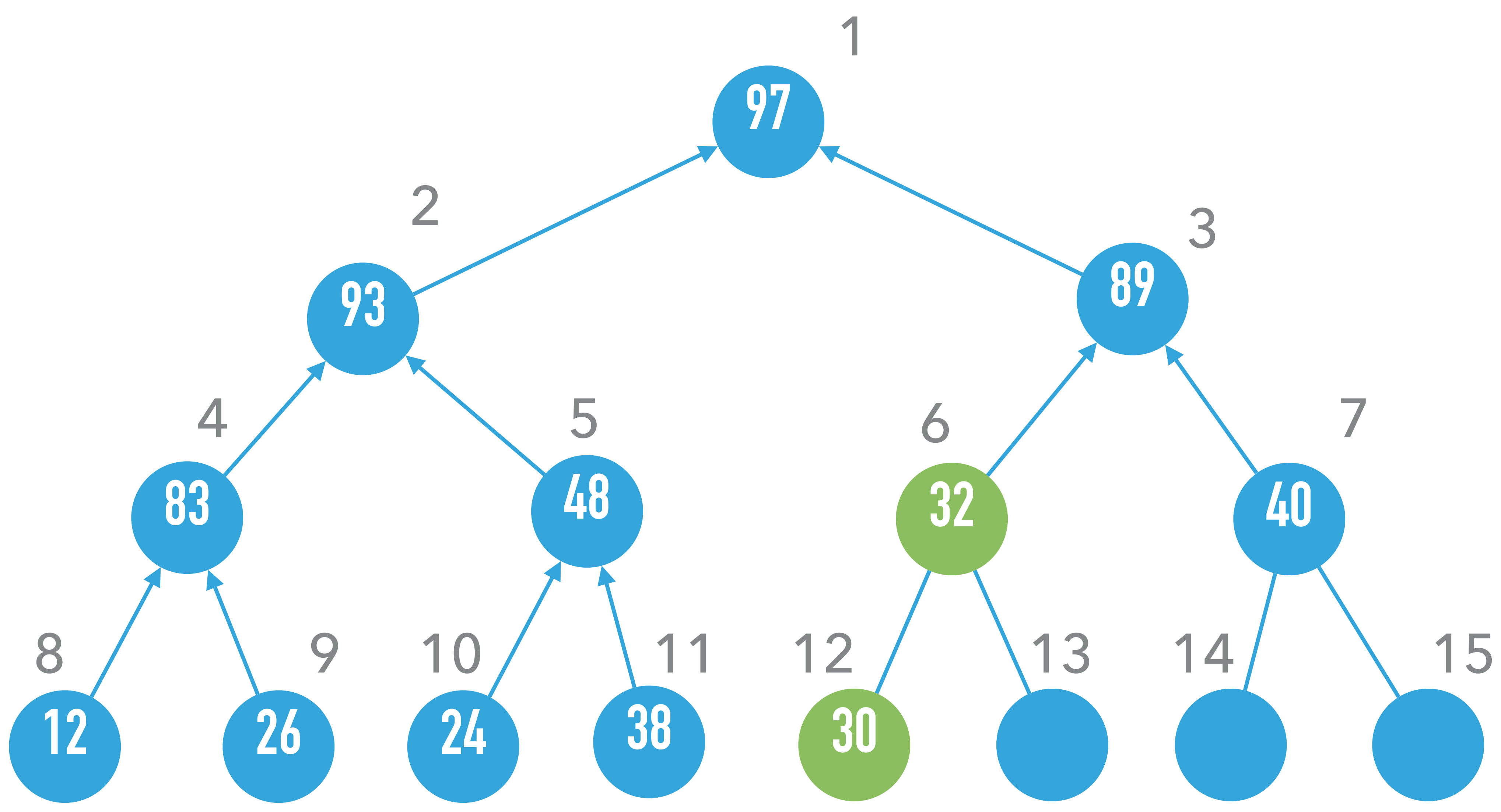
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38	30			



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

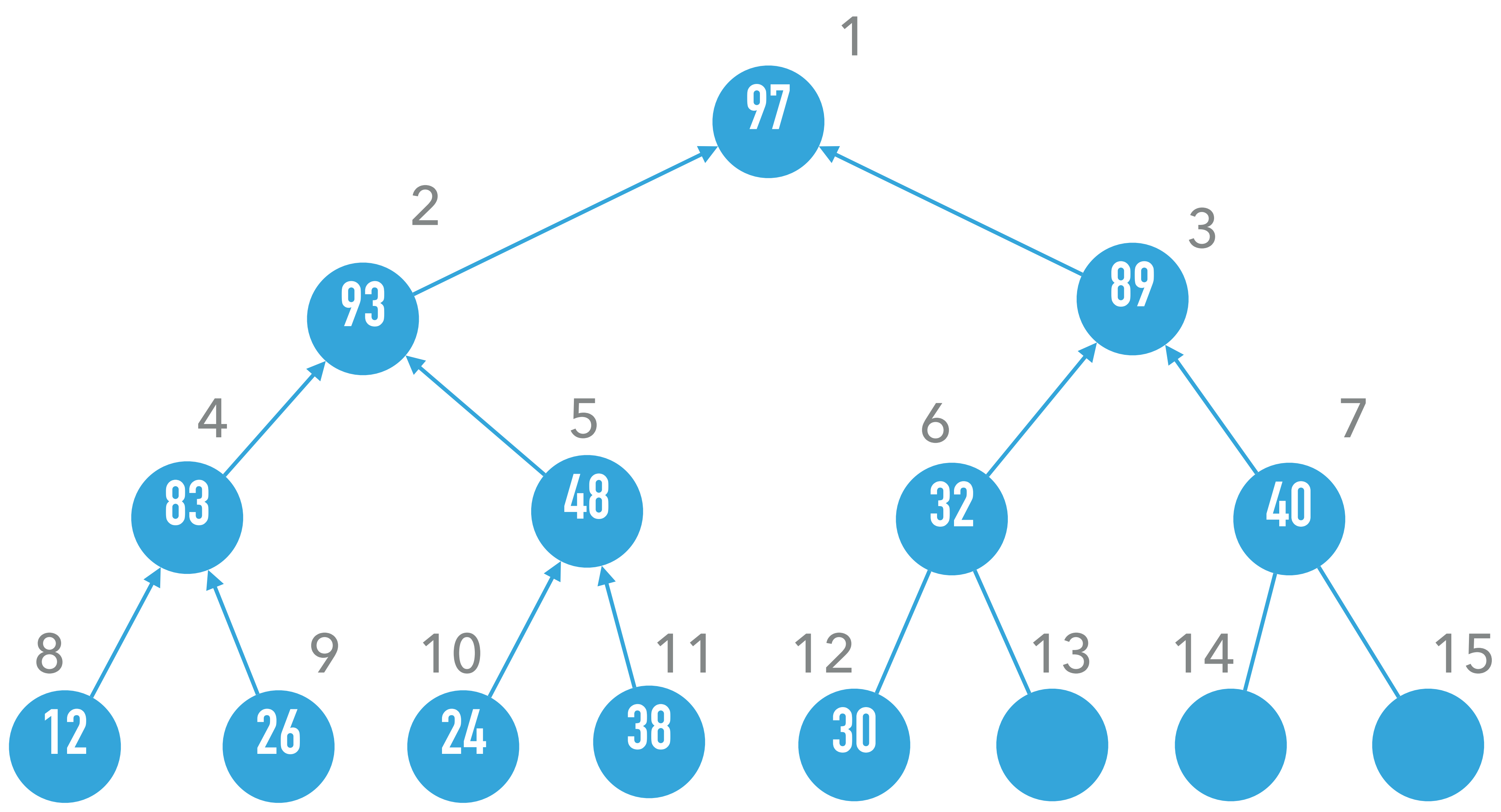
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38	30			



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

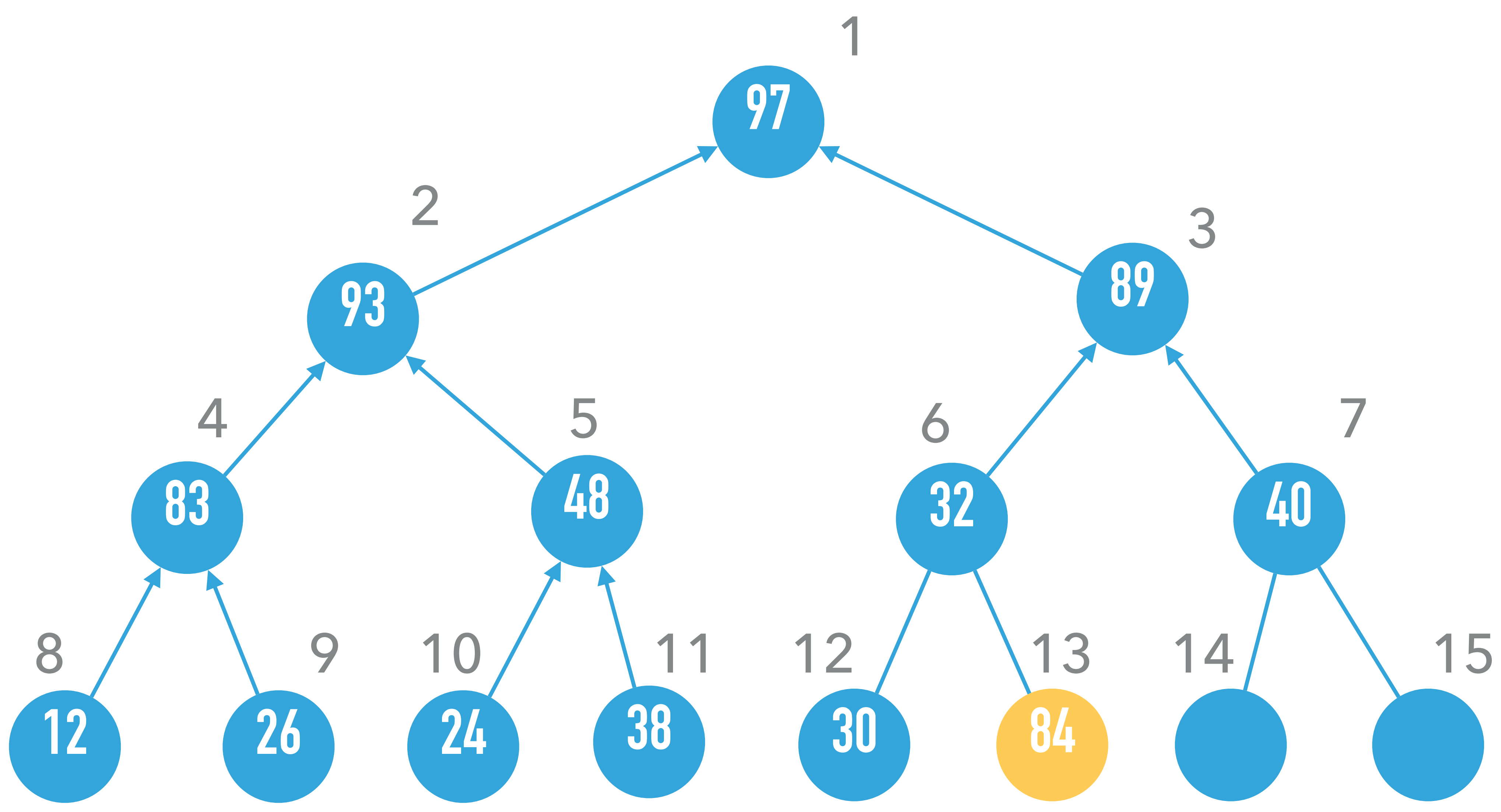
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38	30	84		



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

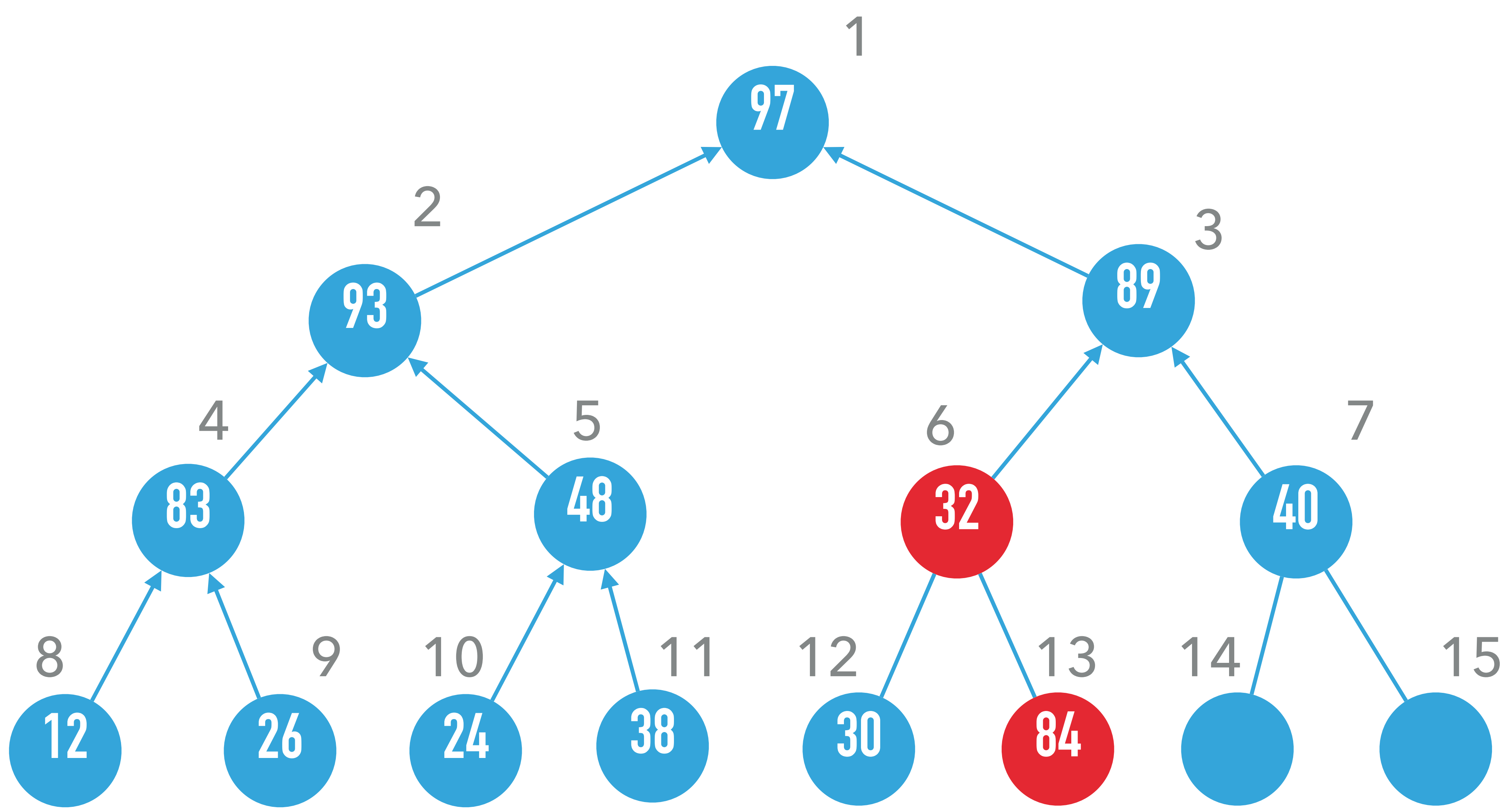
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	32	40	12	26	24	38	30	84		



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

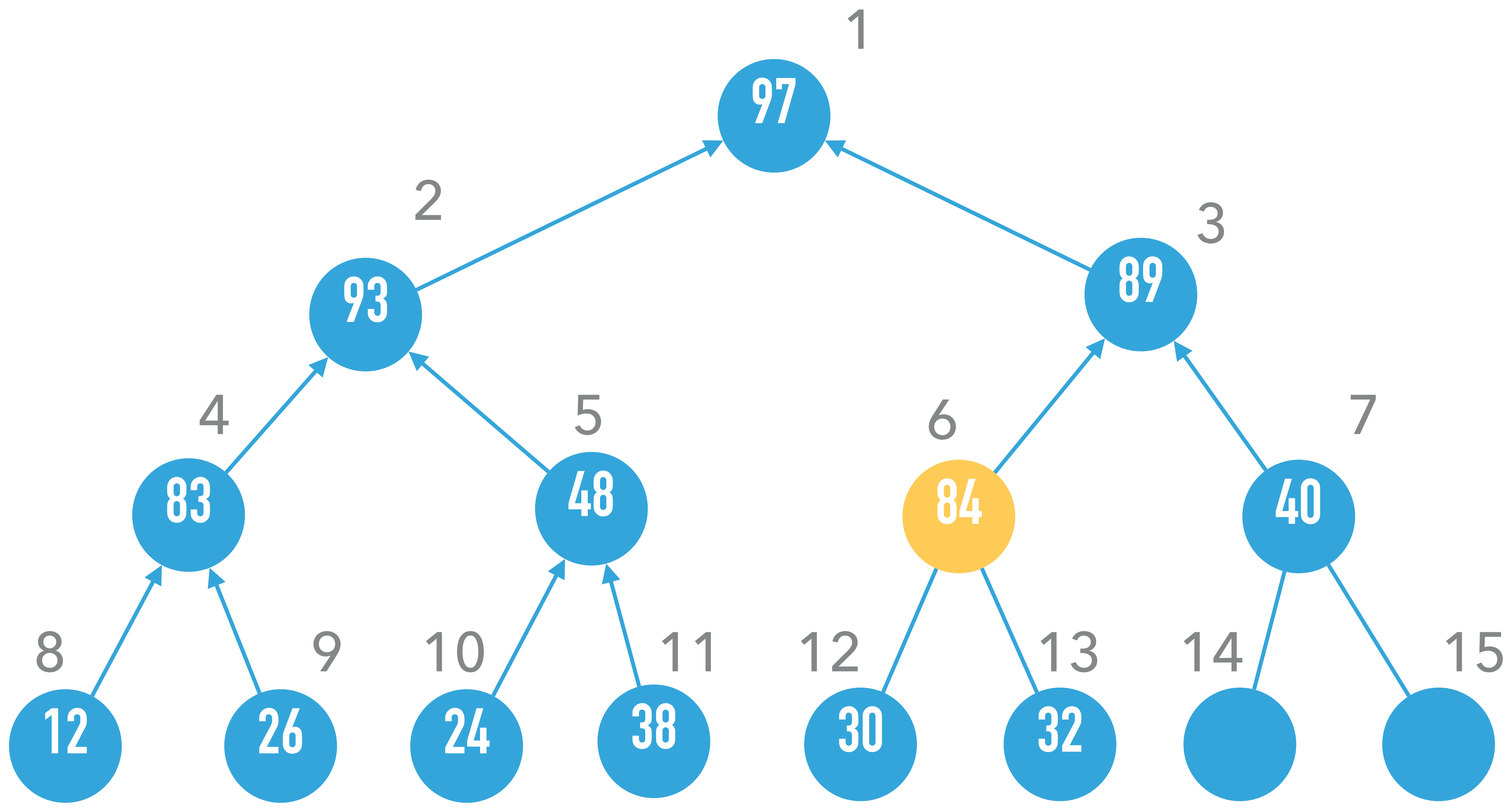
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	84	40	12	26	24	38	30	32		



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

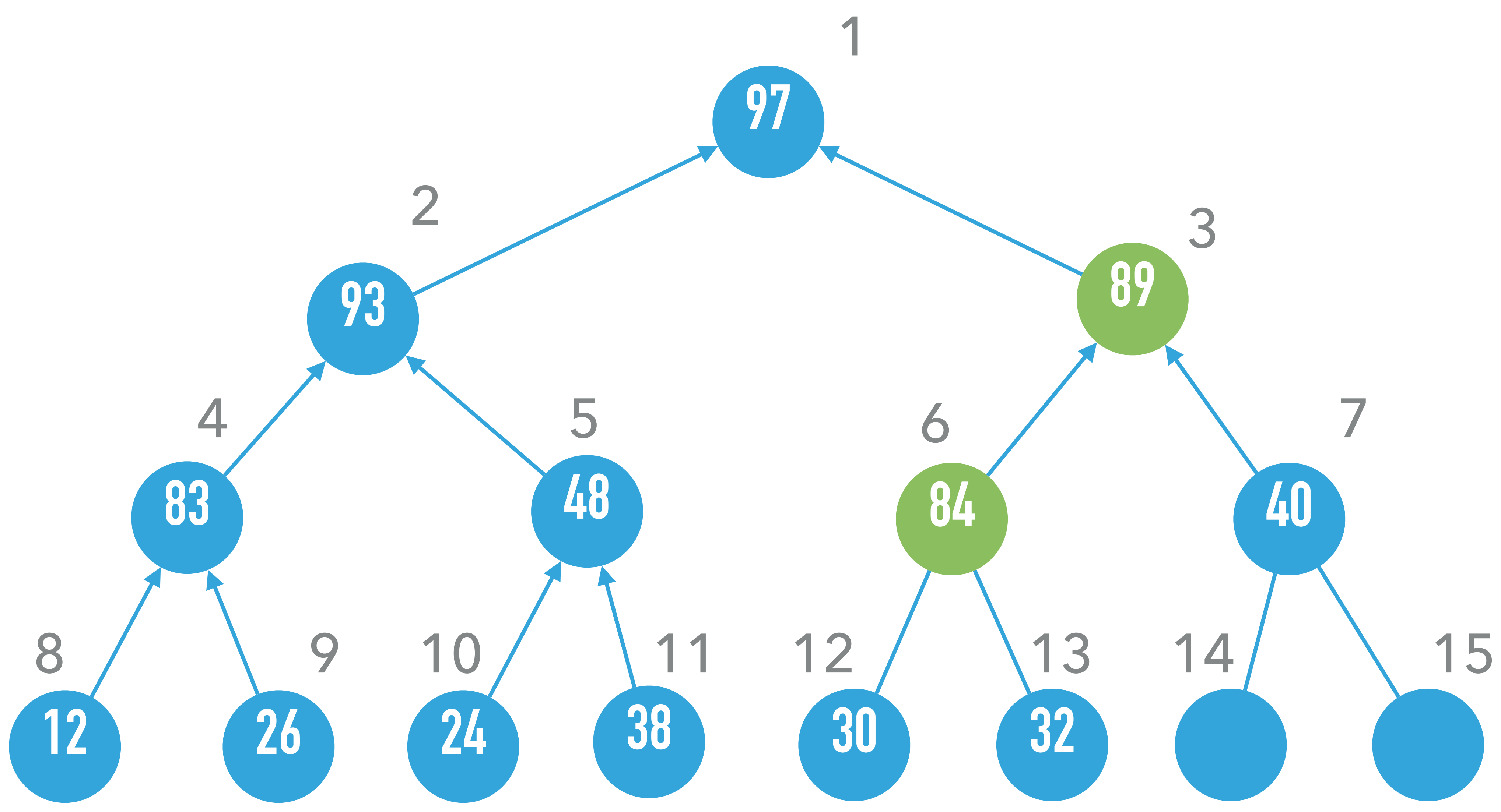
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	84	40	12	26	24	38	30	32		



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

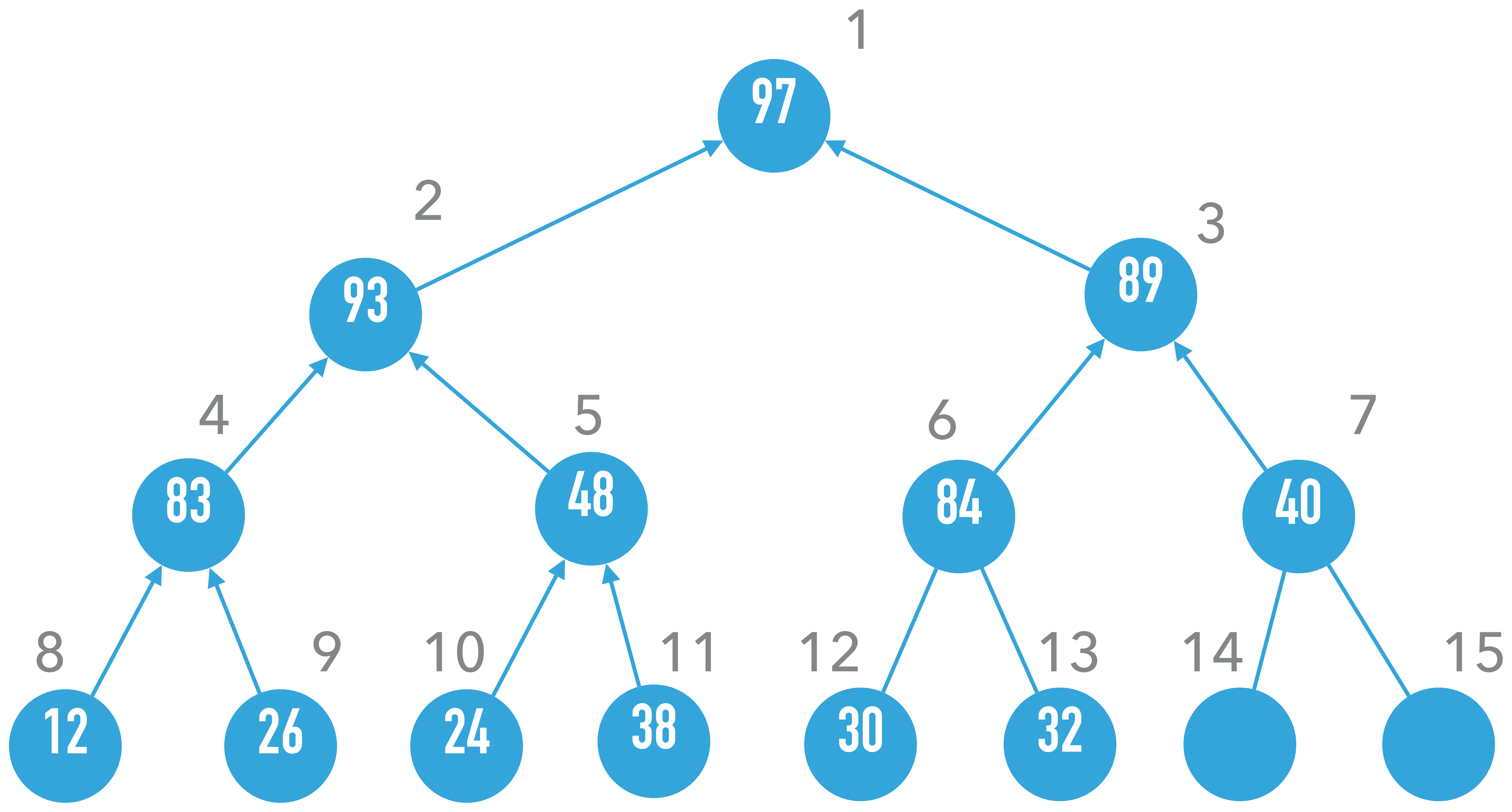
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.4 HEAPS, ajouter des clés

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	97	93	89	83	48	84	40	12	26	24	38	30	32		



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

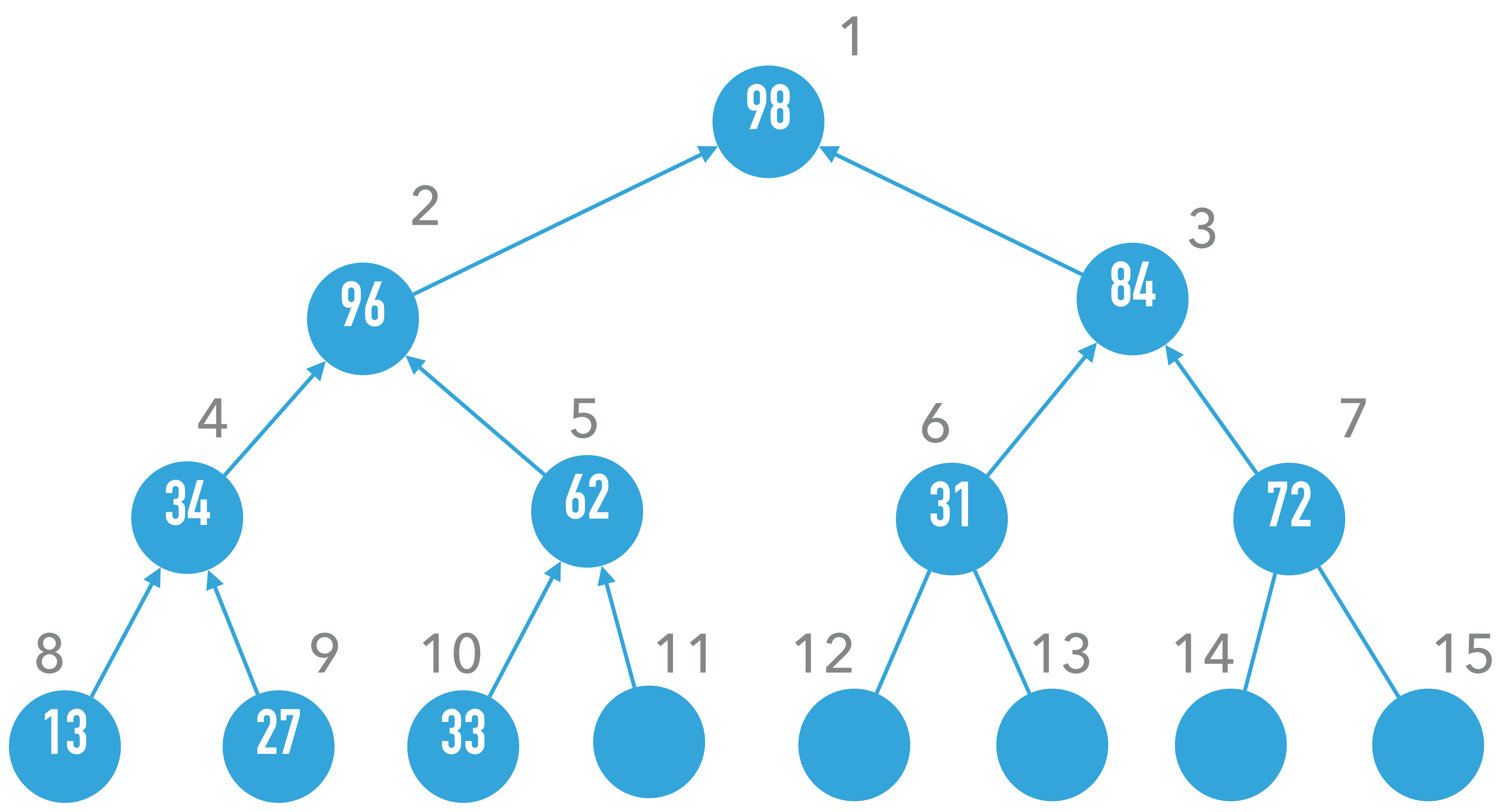
AJOUTER 48

AJOUTER 30

AJOUTER 84

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	98	96	84	34	62	31	72	13	27	33					



Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

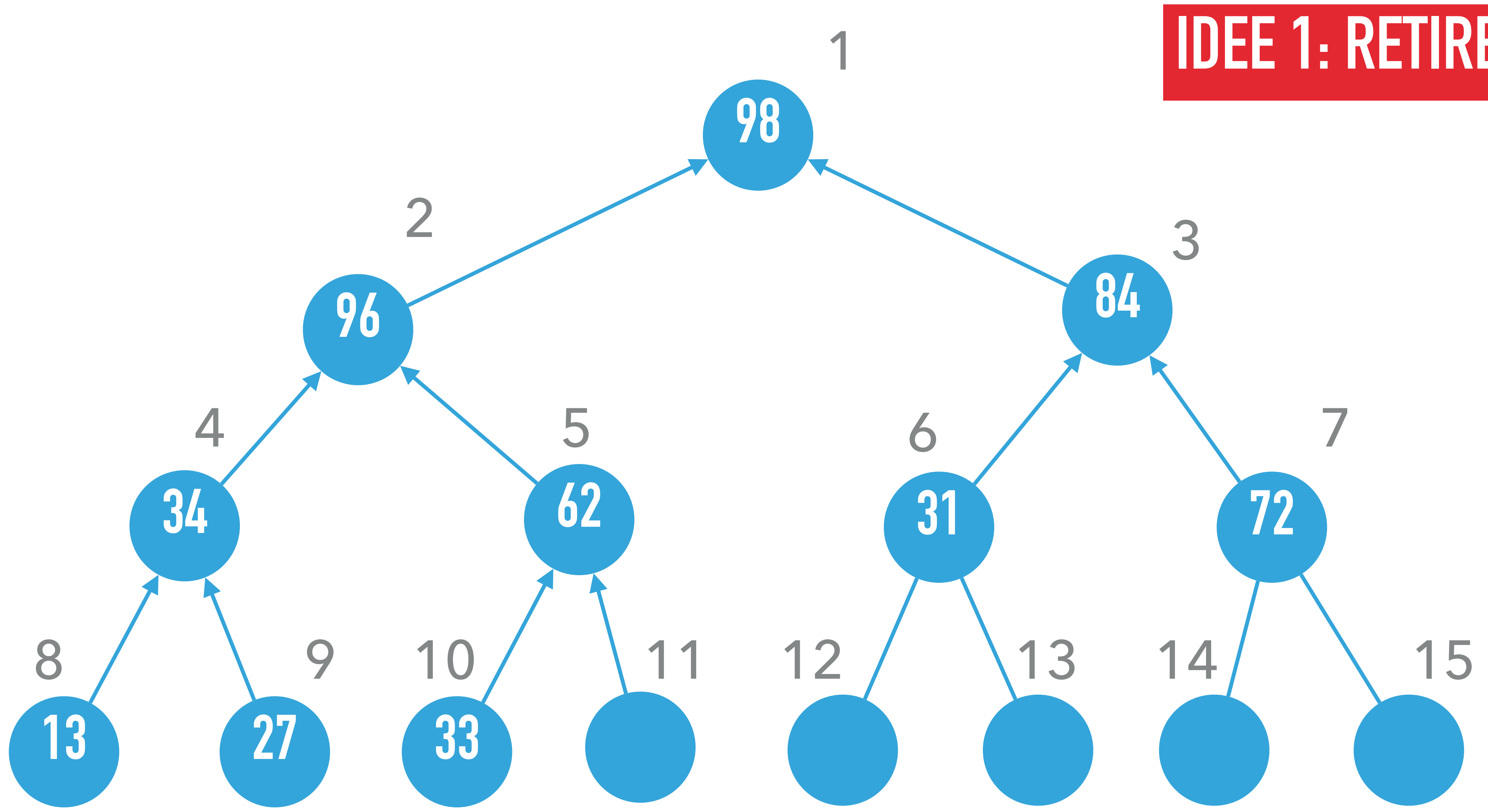
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	98	96	84	34	62	31	72	13	27	33					



IDEE 1: RETIRER LE MAX ET CORRIGER EN PARTANT DU HAUT

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

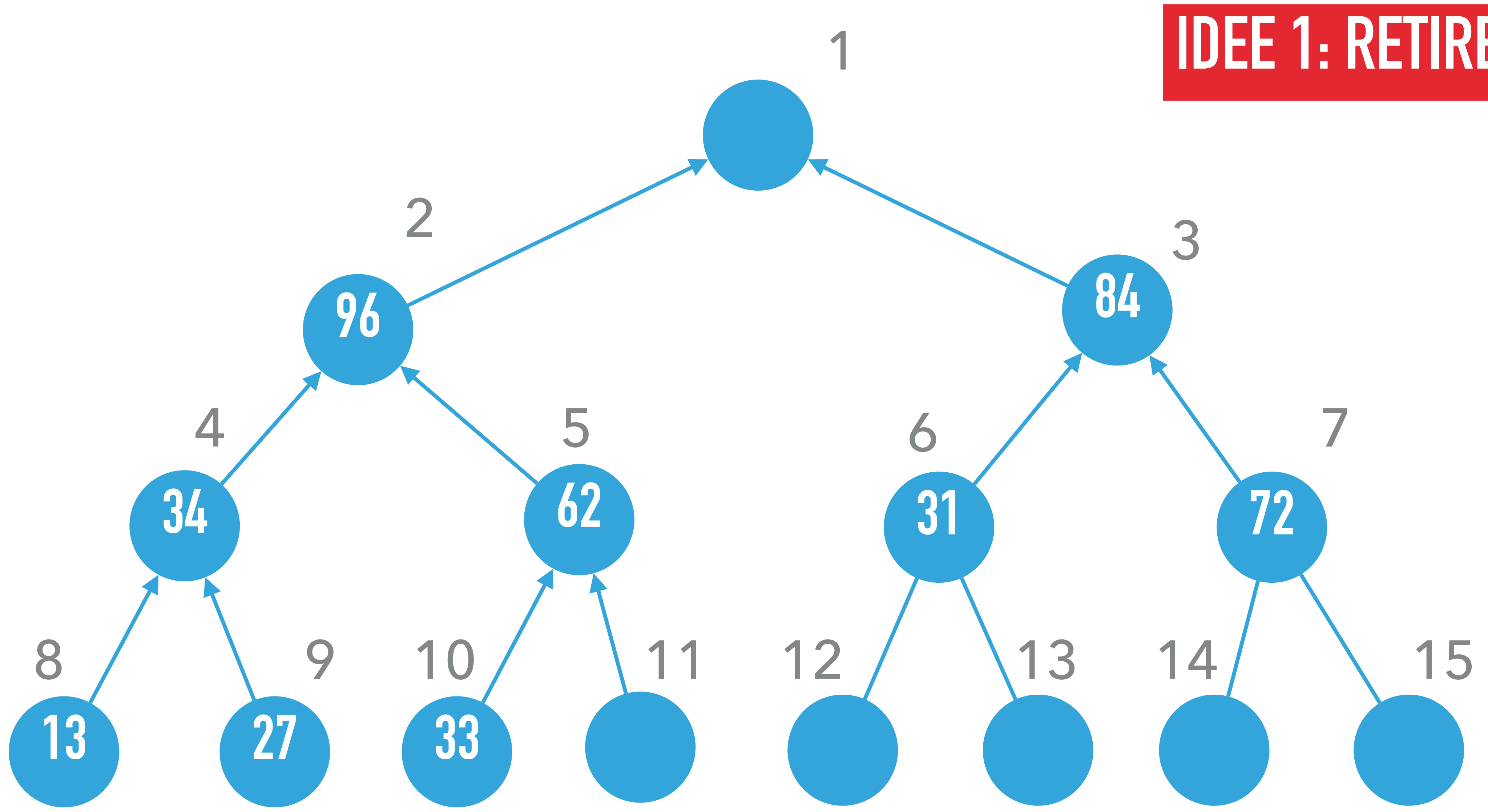
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V		96	84	34	62	31	72	13	27	33					



IDEE 1: RETIRER LE MAX ET CORRIGER EN PARTANT DU HAUT

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

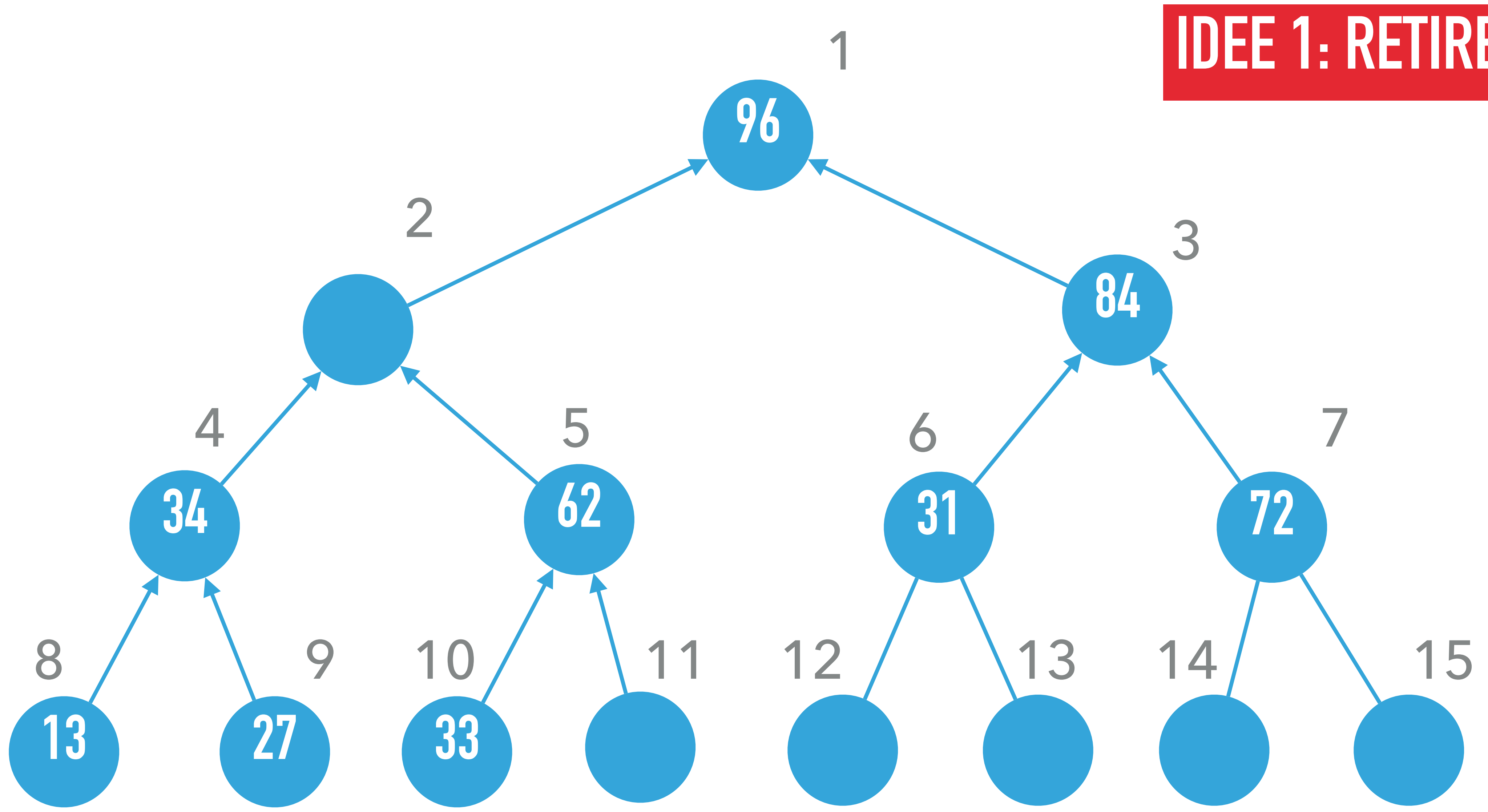
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96		84	34	62	31	72	13	27	33					



IDEE 1: RETIRER LE MAX ET CORRIGER EN PARTANT DU HAUT

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

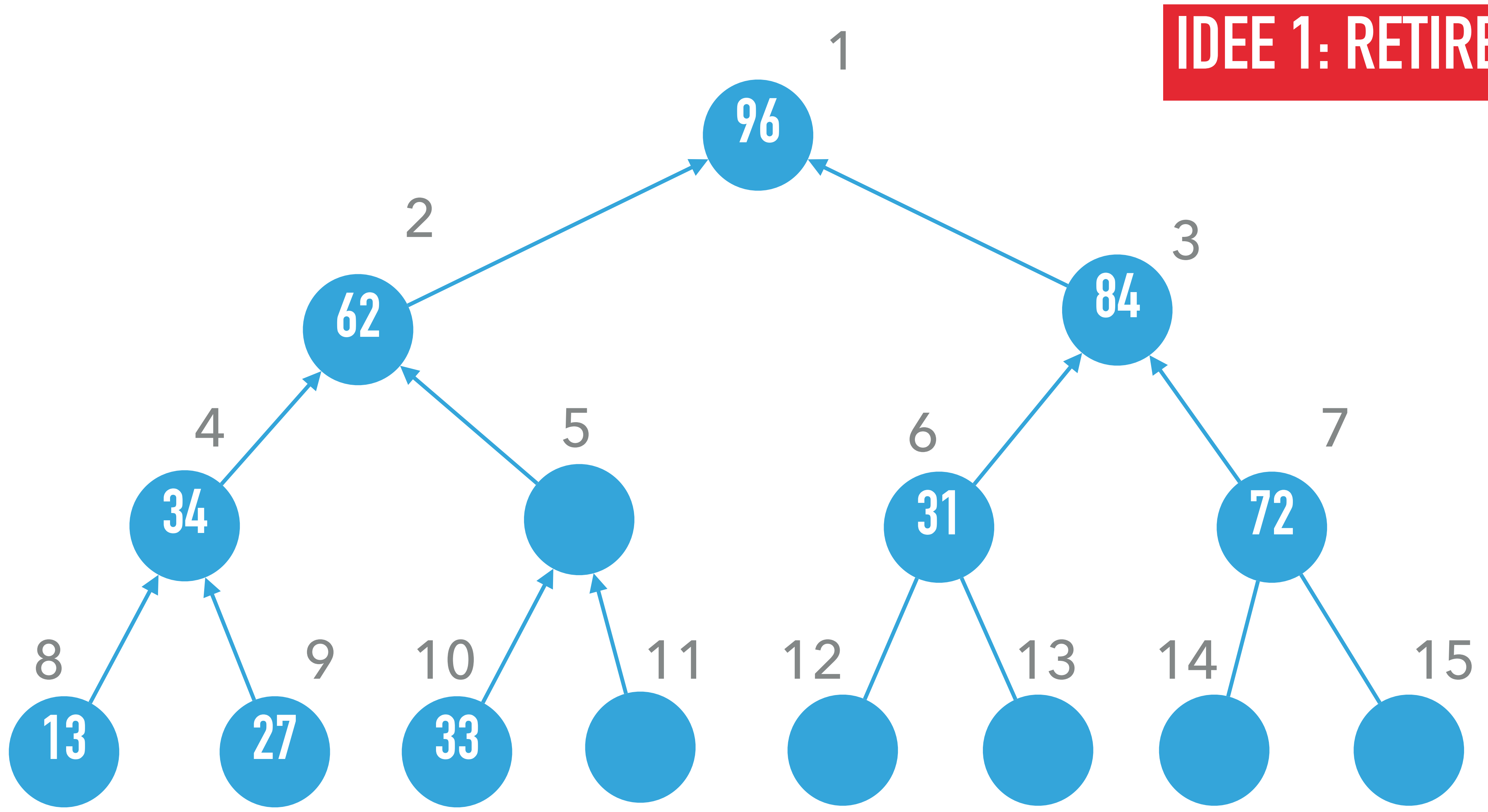
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	62	84	34		31	72	13	27	33					



IDEE 1: RETIRER LE MAX ET CORRIGER EN PARTANT DU HAUT

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

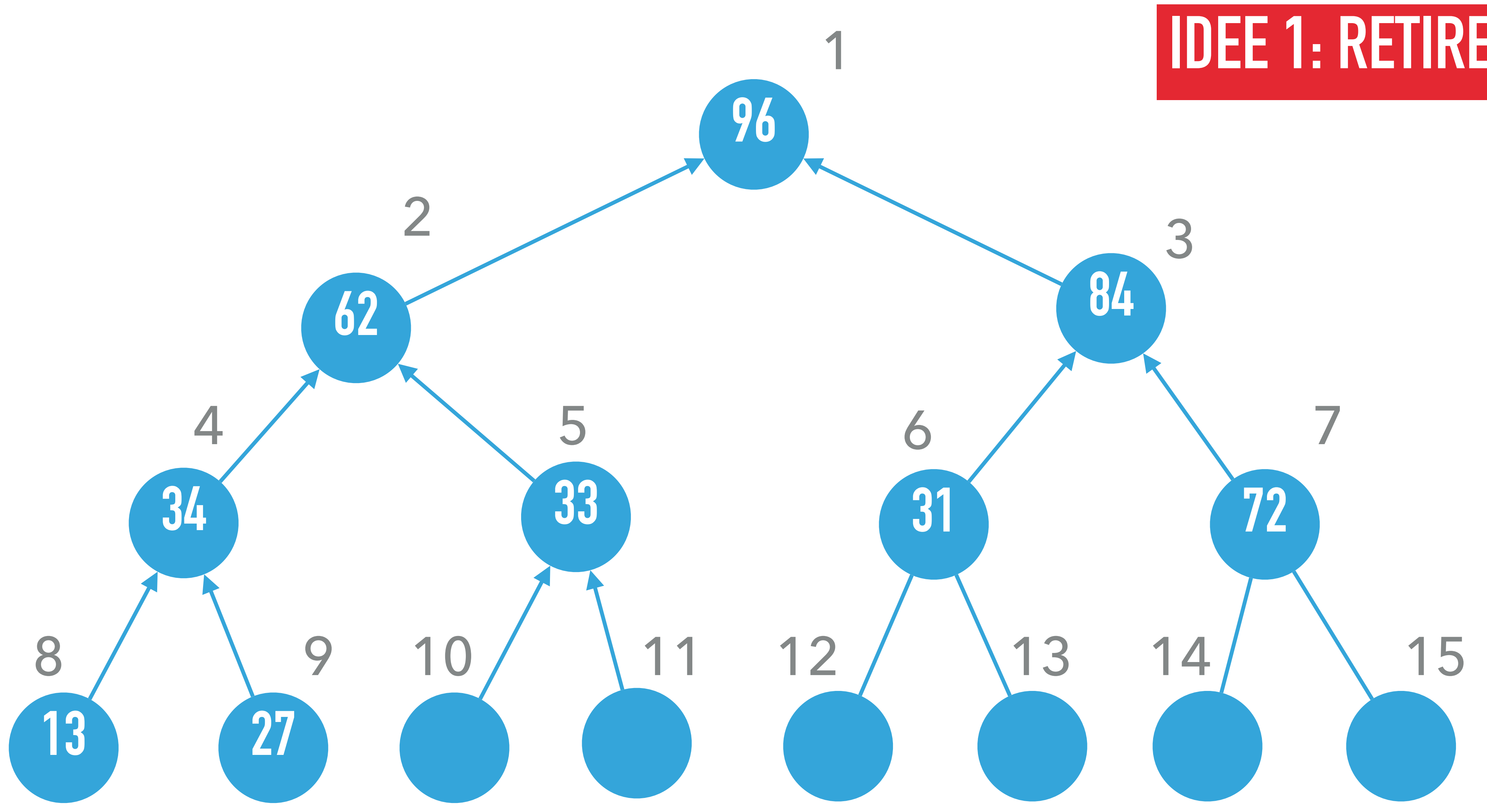
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	62	84	34	33	31	72	13	27						



IDEE 1: RETIRER LE MAX ET CORRIGER EN PARTANT DU HAUT

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

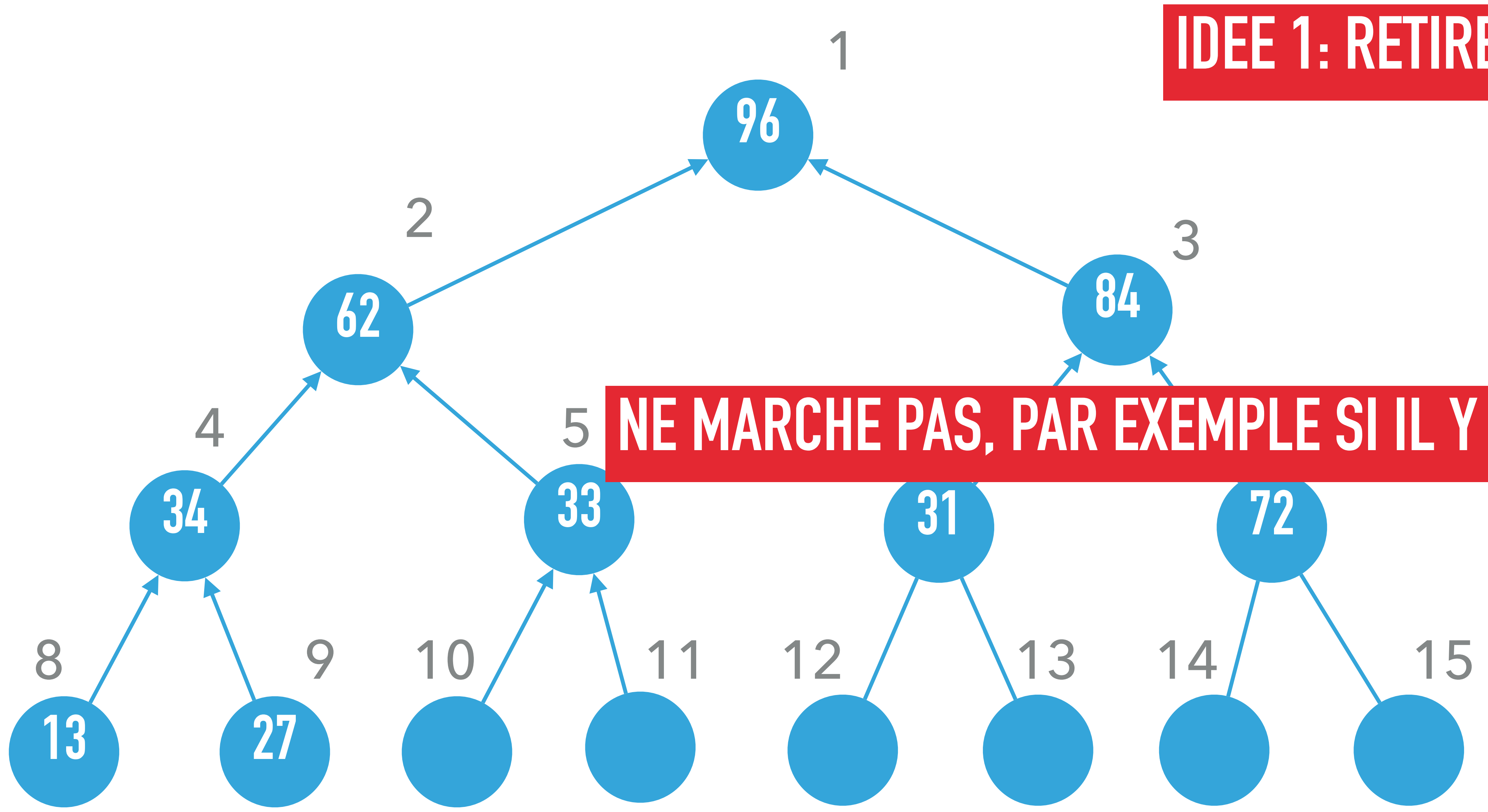
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	62	84	34	33	31	72	13	27						



IDEE 1: RETIRER LE MAX ET CORRIGER EN PARTANT DU HAUT

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

NE MARCHE PAS, PAR EXEMPLE SI IL Y A DES ENTREES EN 11 ET 12

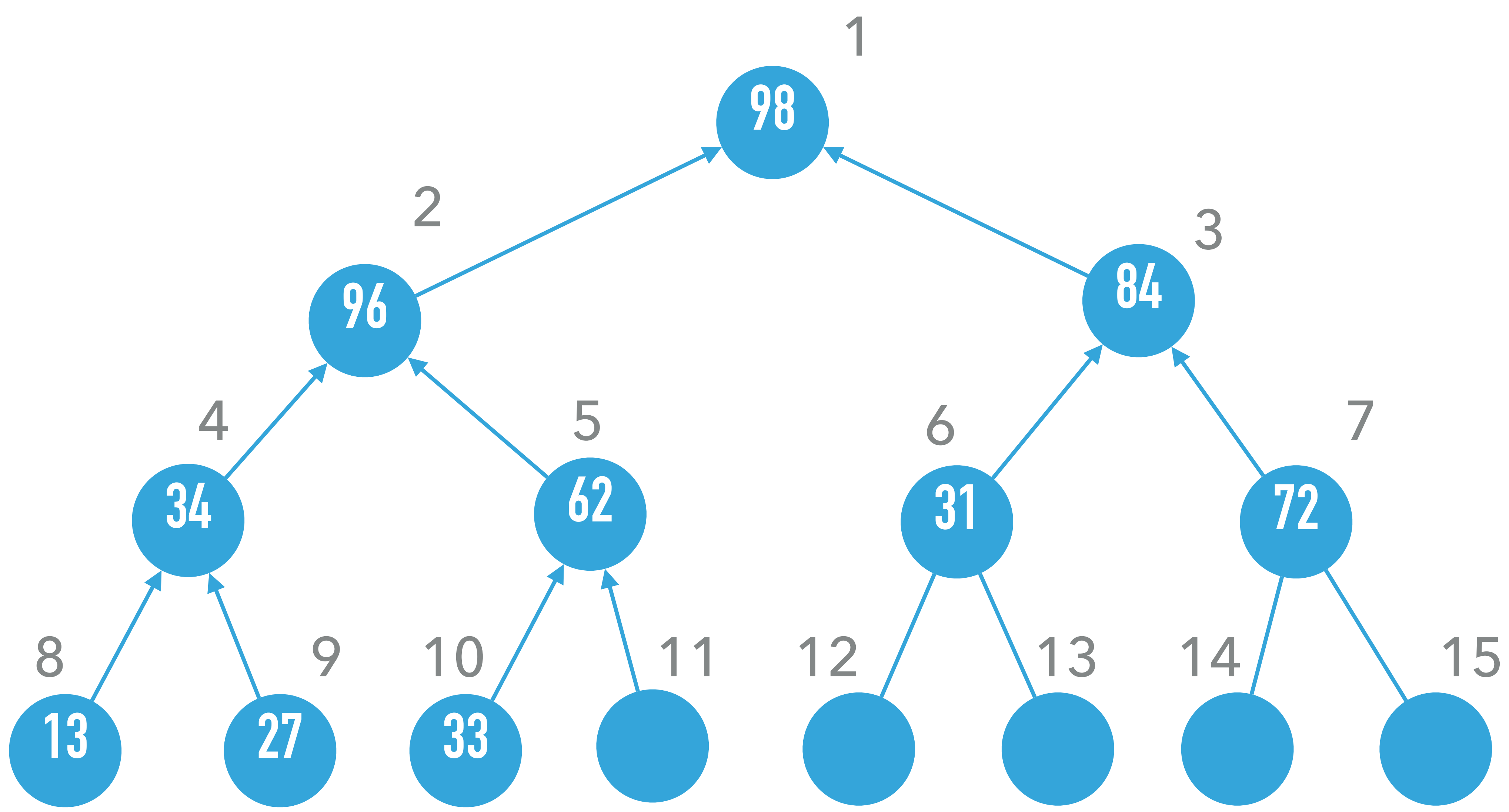
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	98	96	84	34	62	31	72	13	27	33					



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

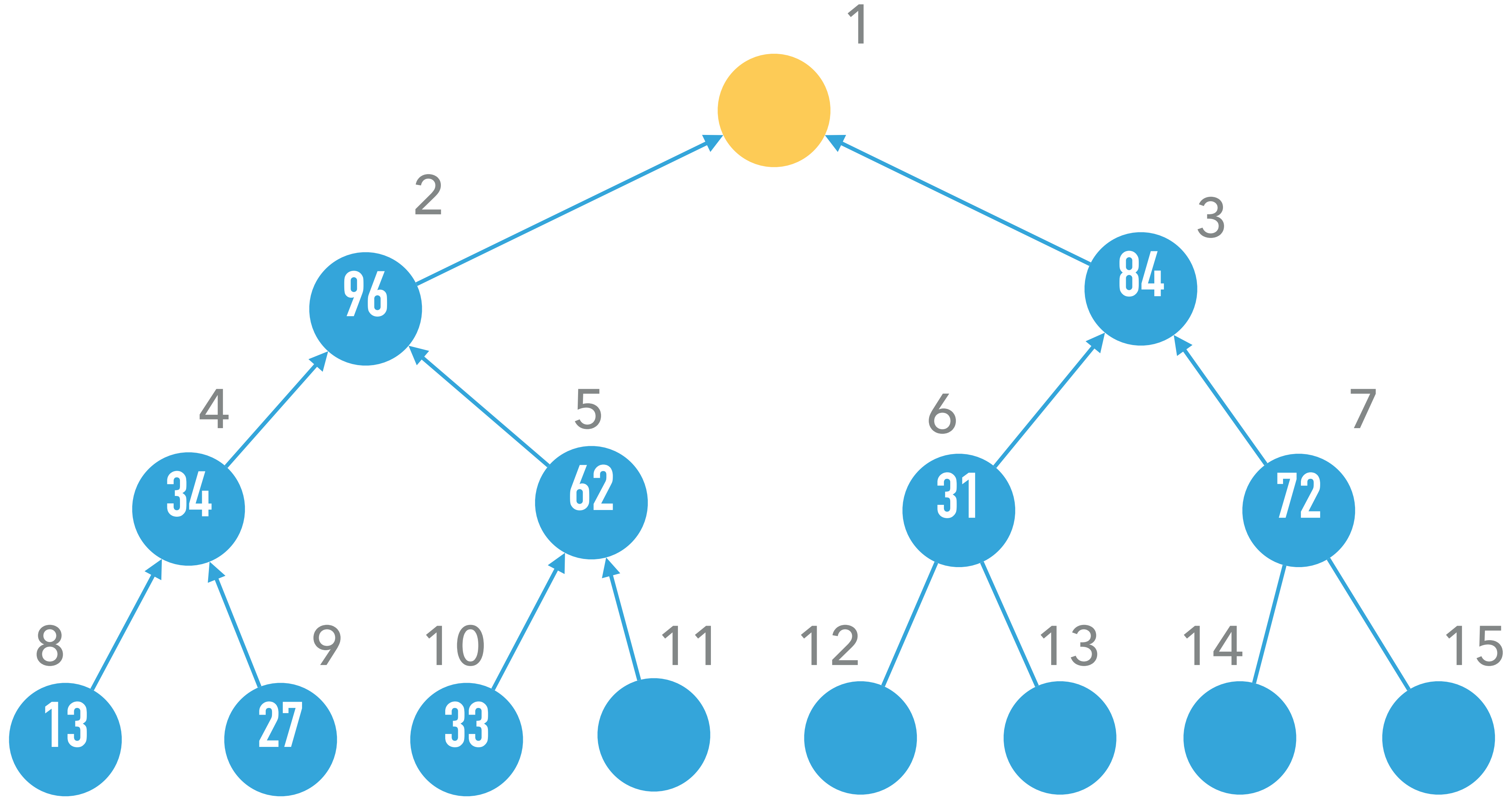
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V		96	84	34	62	31	72	13	27	33					



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

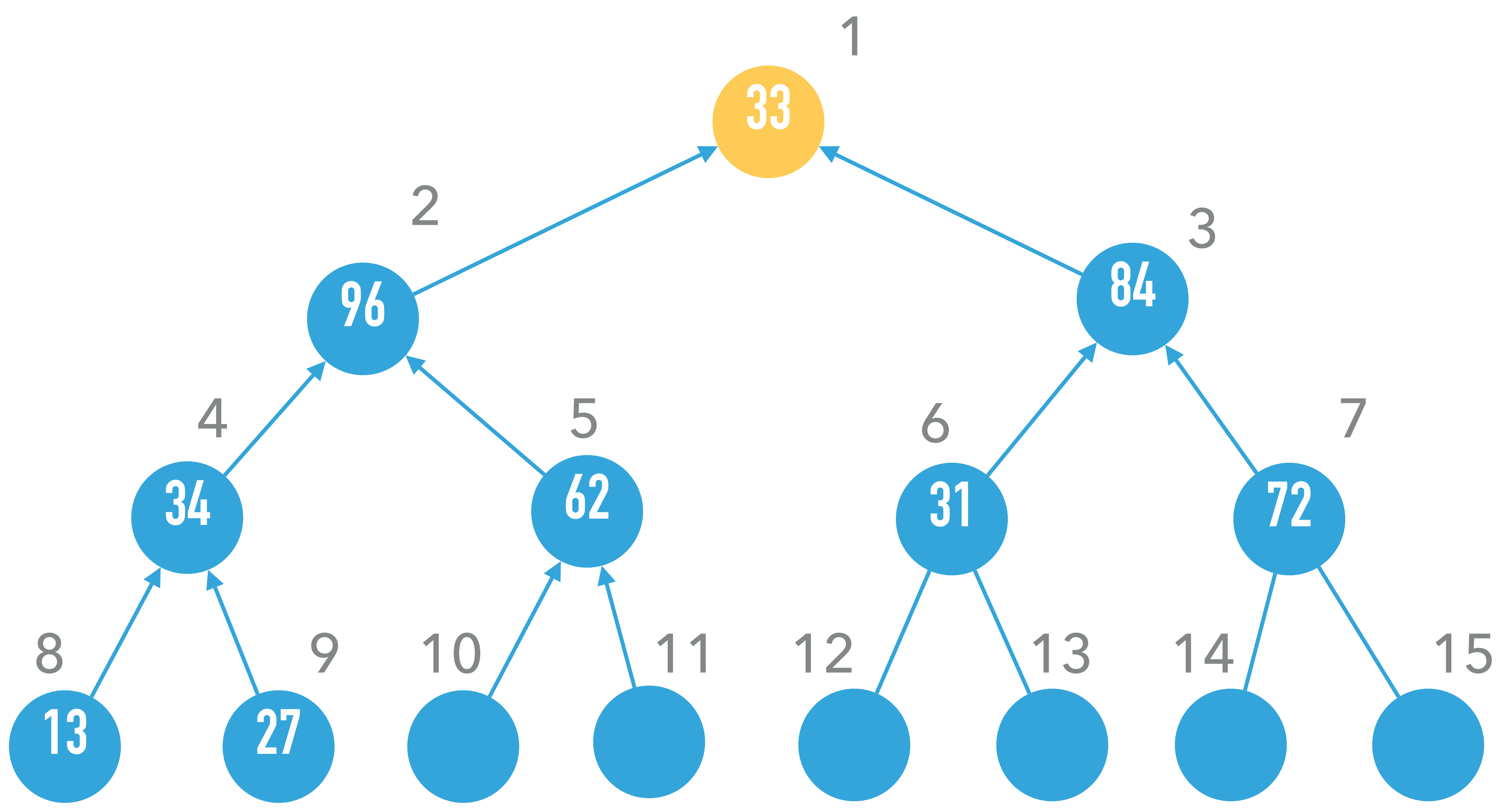
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	33	96	84	34	62	31	72	13	27						



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

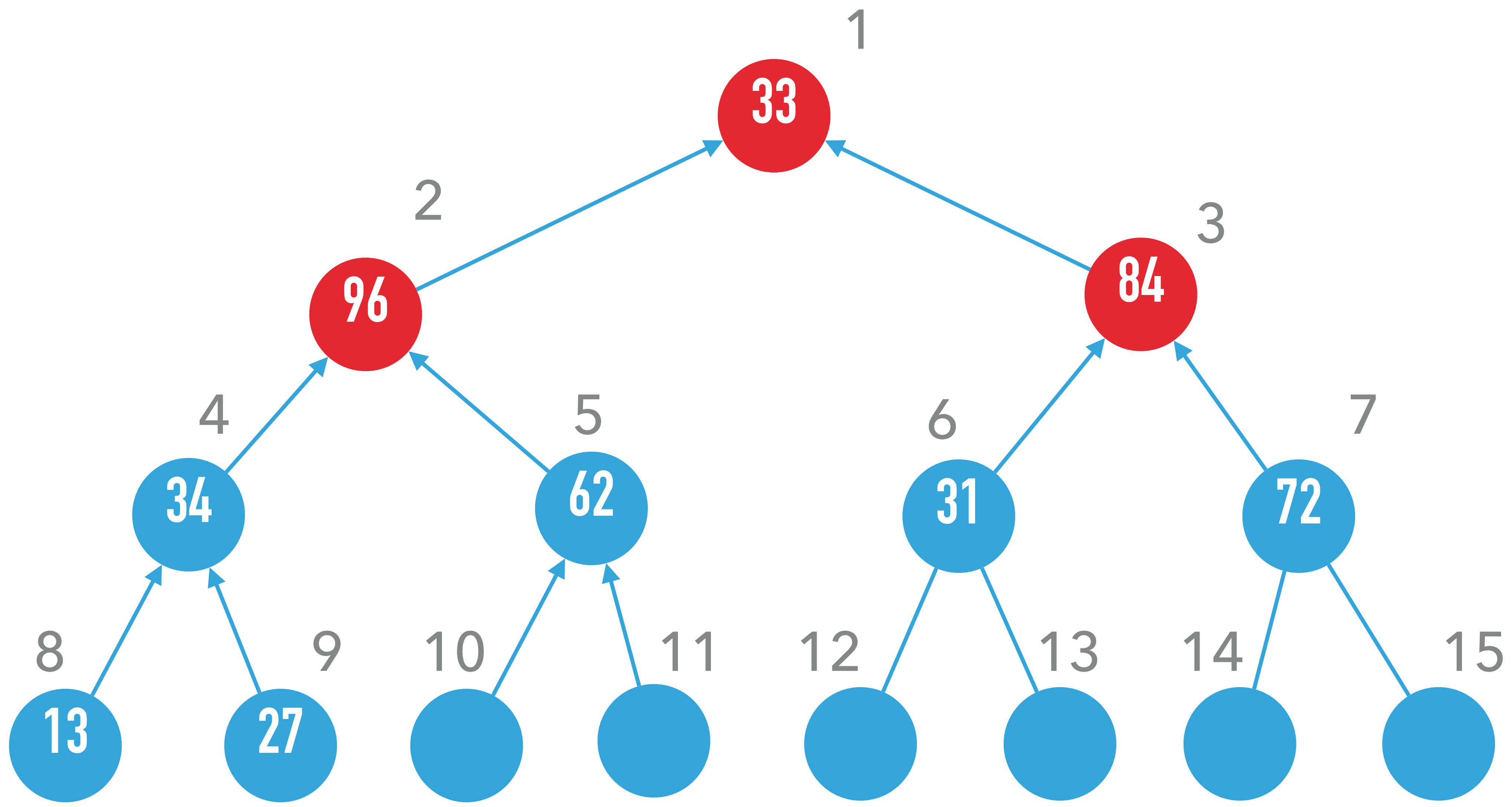
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	33	96	84	34	62	31	72	13	27						



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

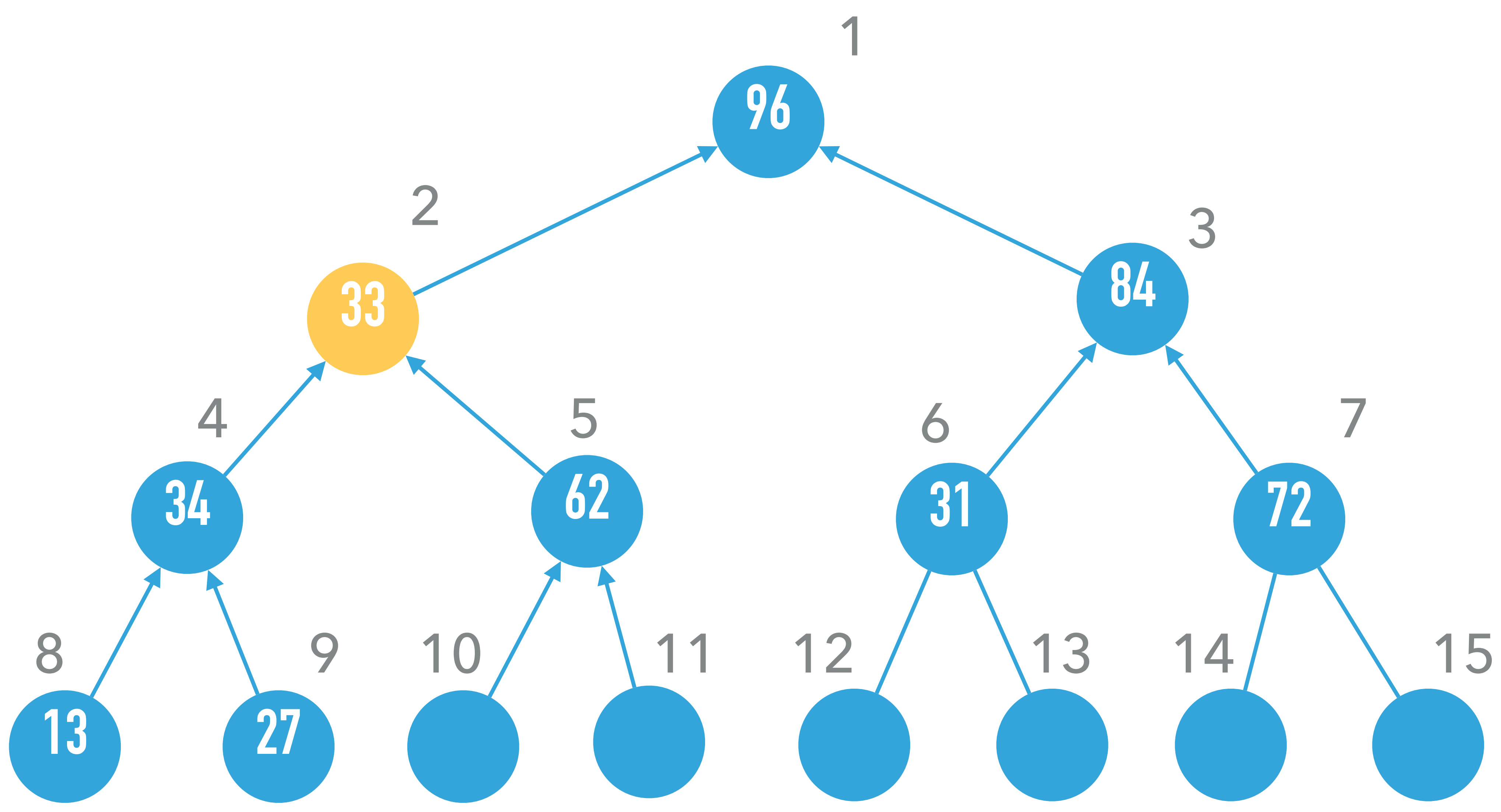
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	33	84	34	62	31	72	13	27						



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

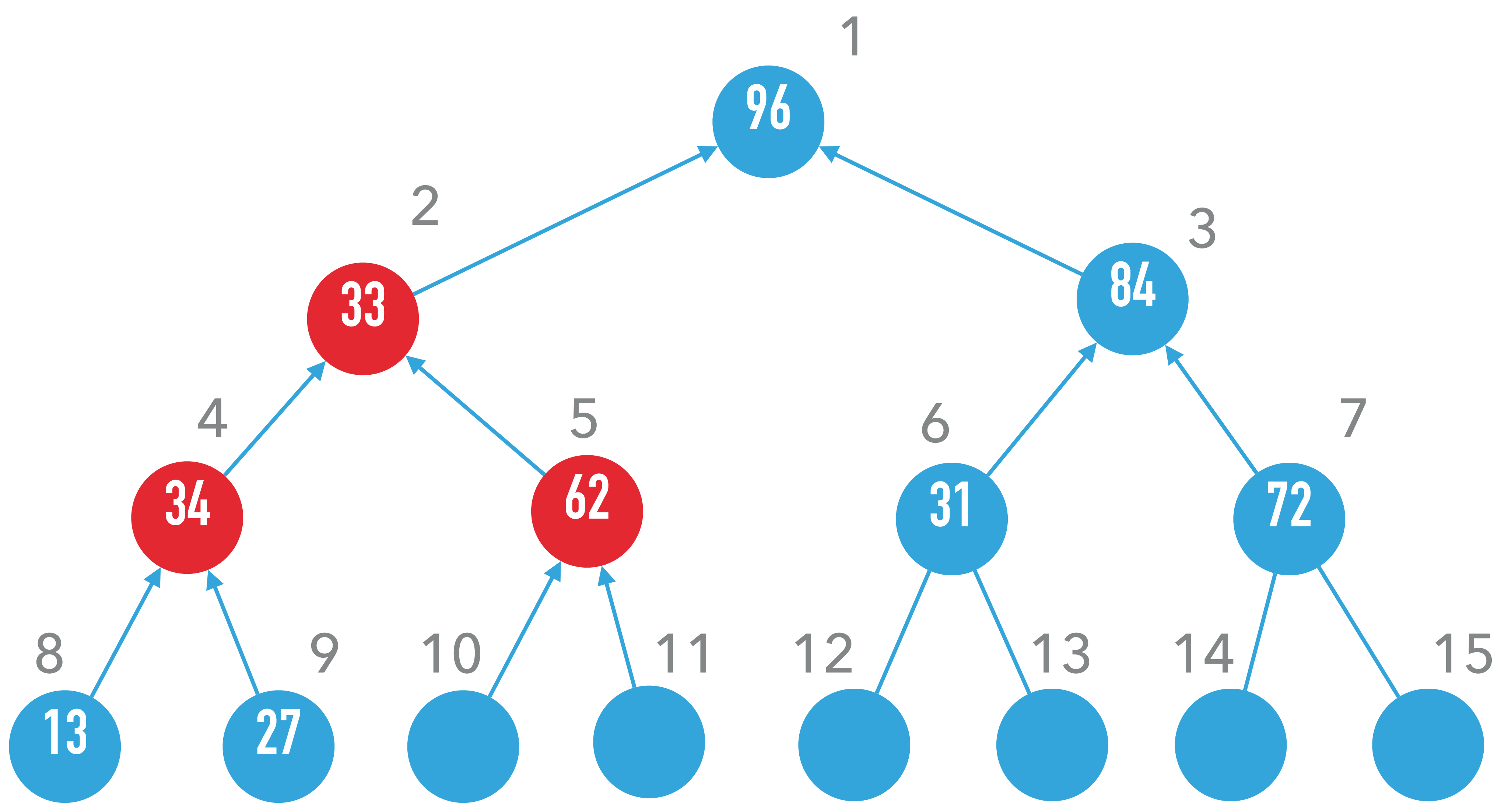
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	33	84	34	62	31	72	13	27						



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

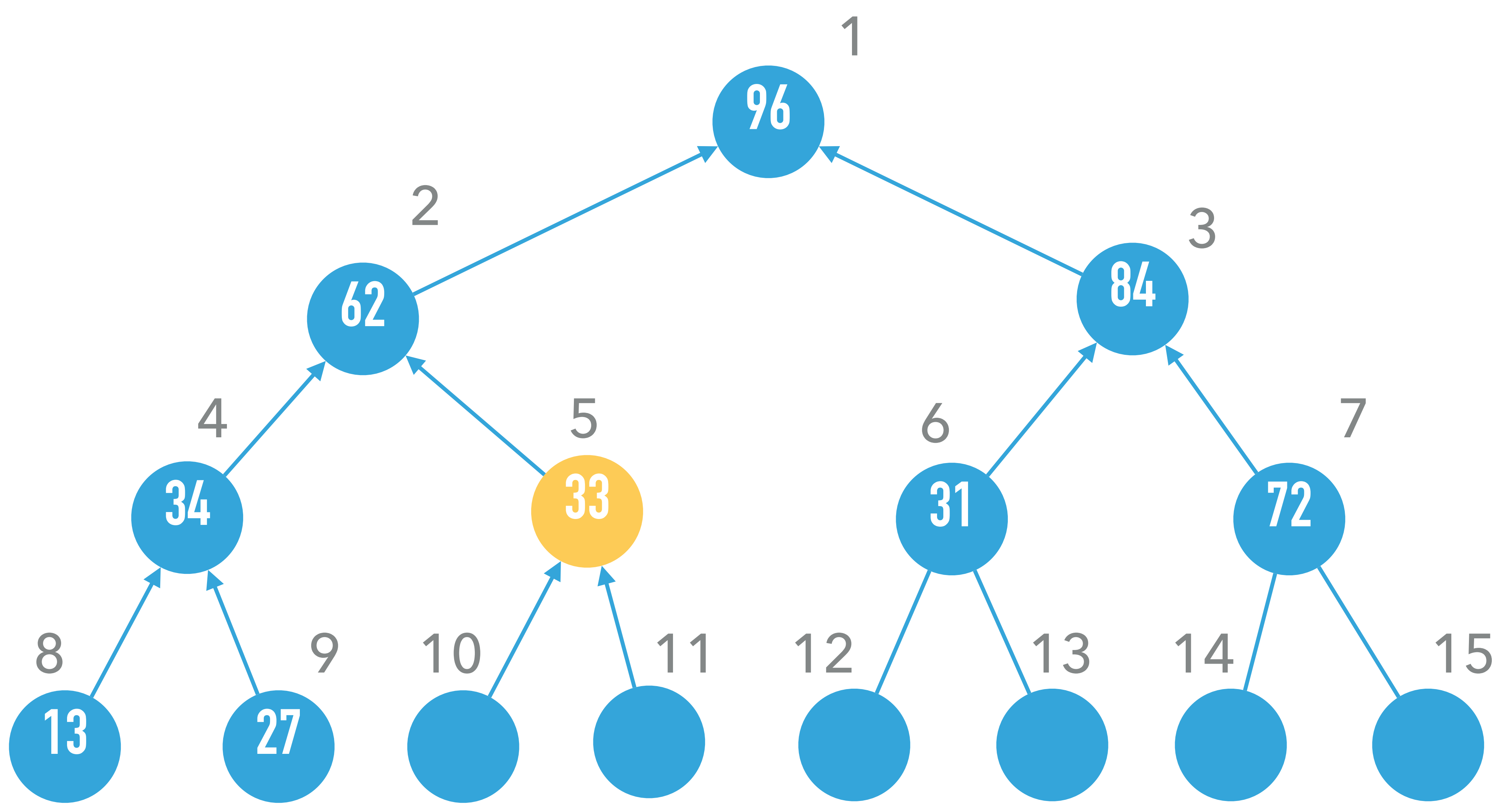
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	62	84	34	33	31	72	13	27						



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

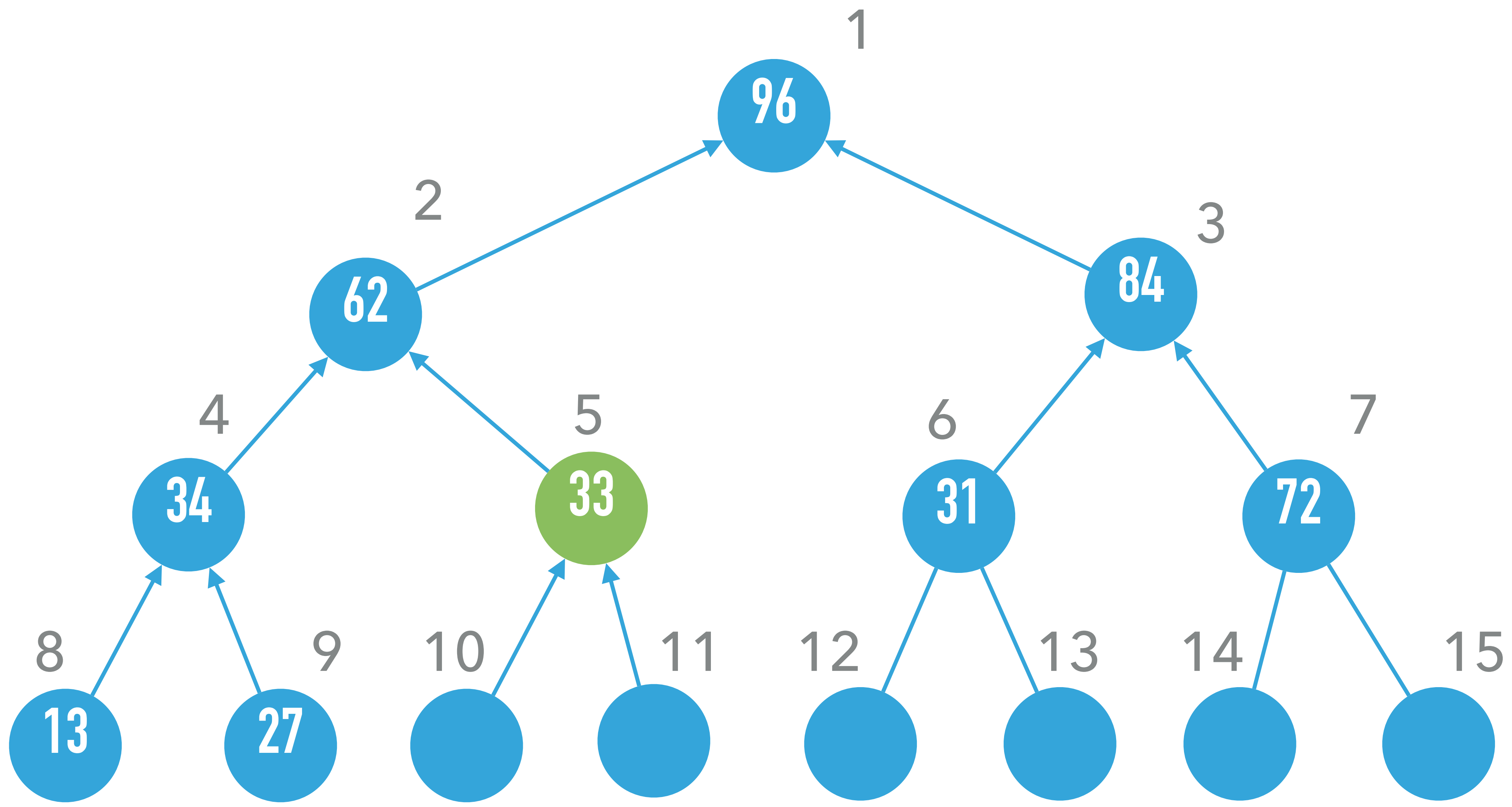
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	62	84	34	33	31	72	13	27						



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

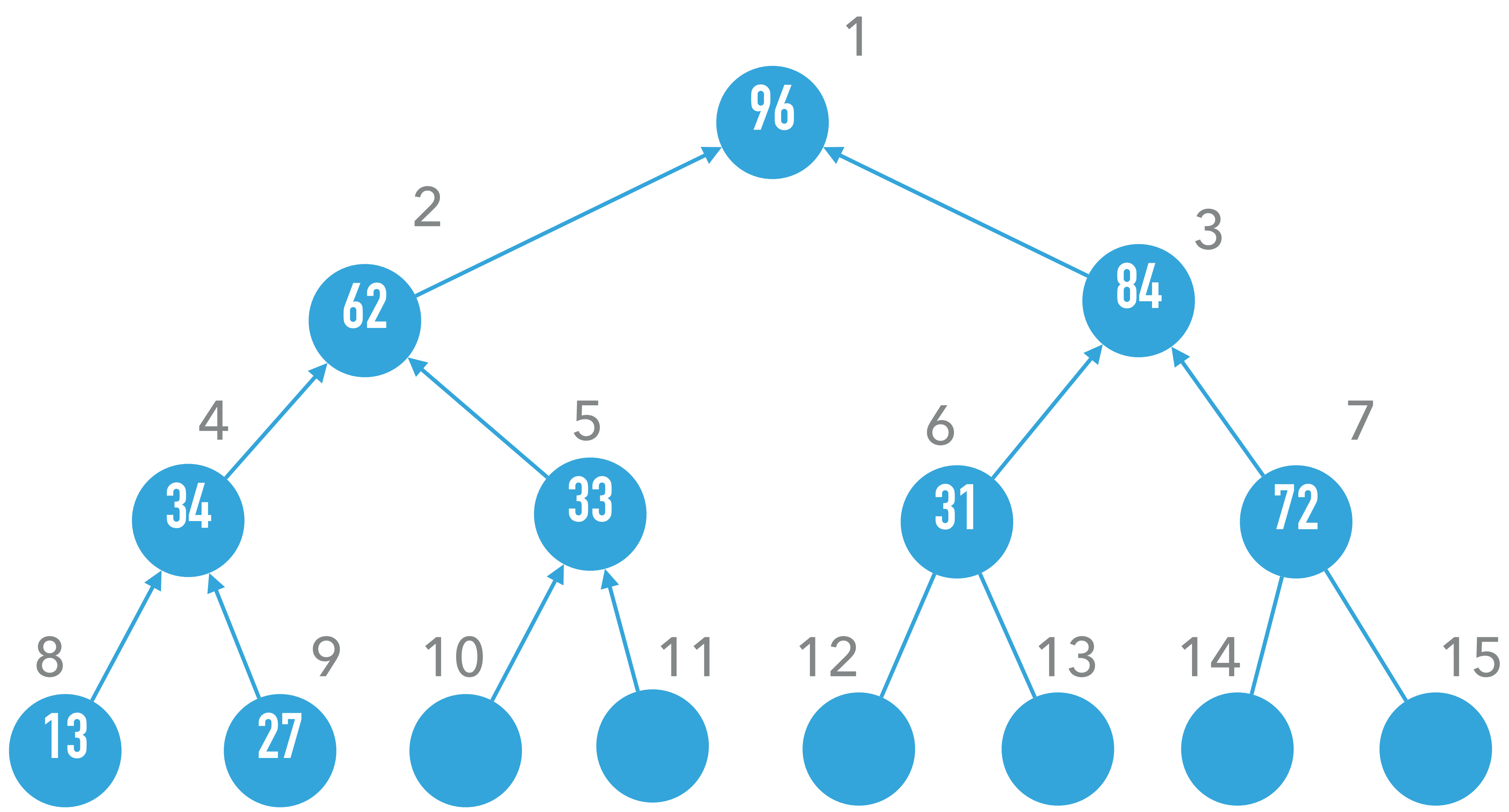
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	96	62	84	34	33	31	72	13	27						



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

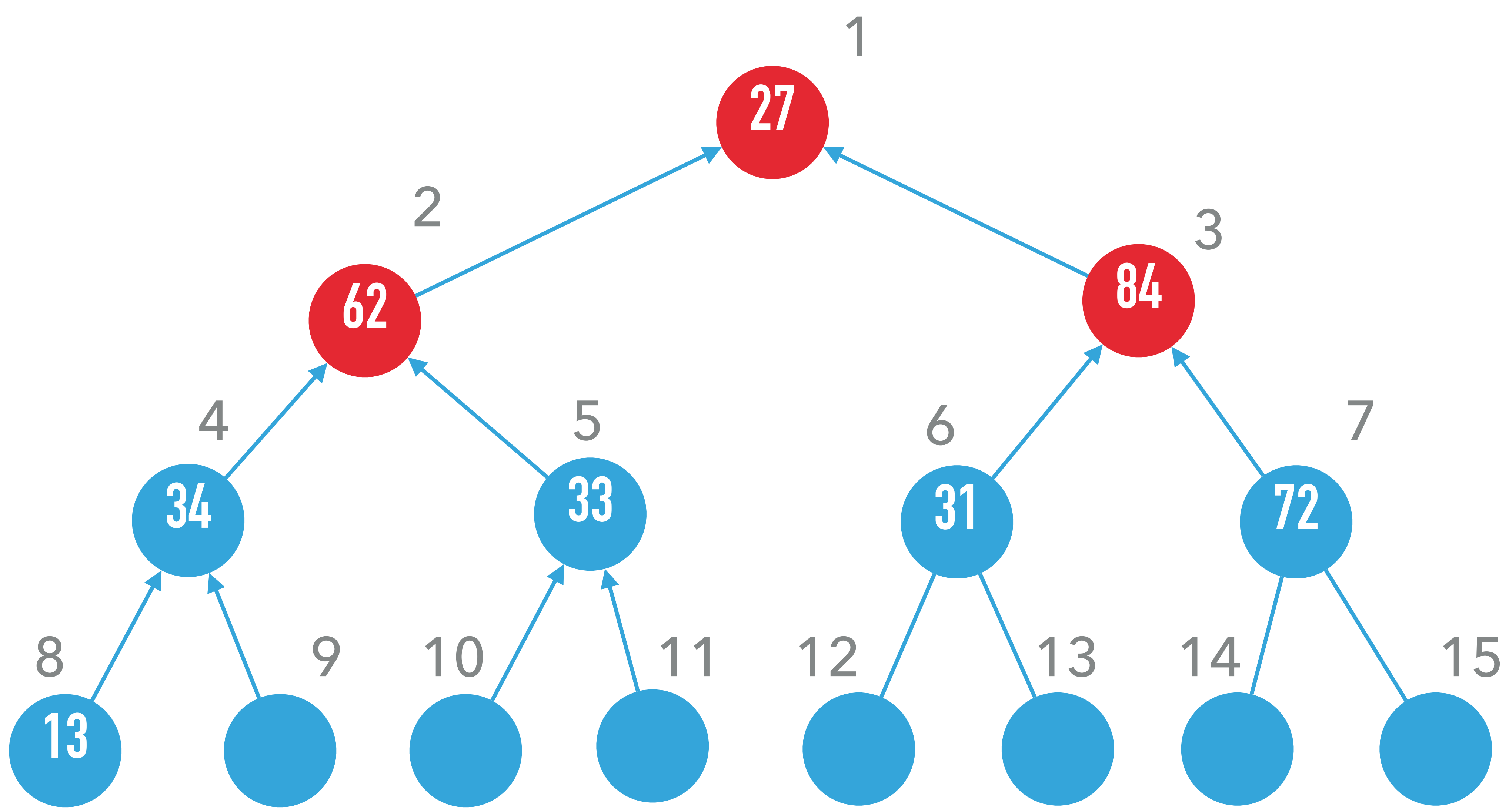
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	27	62	84	34	33	31	72	13							



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

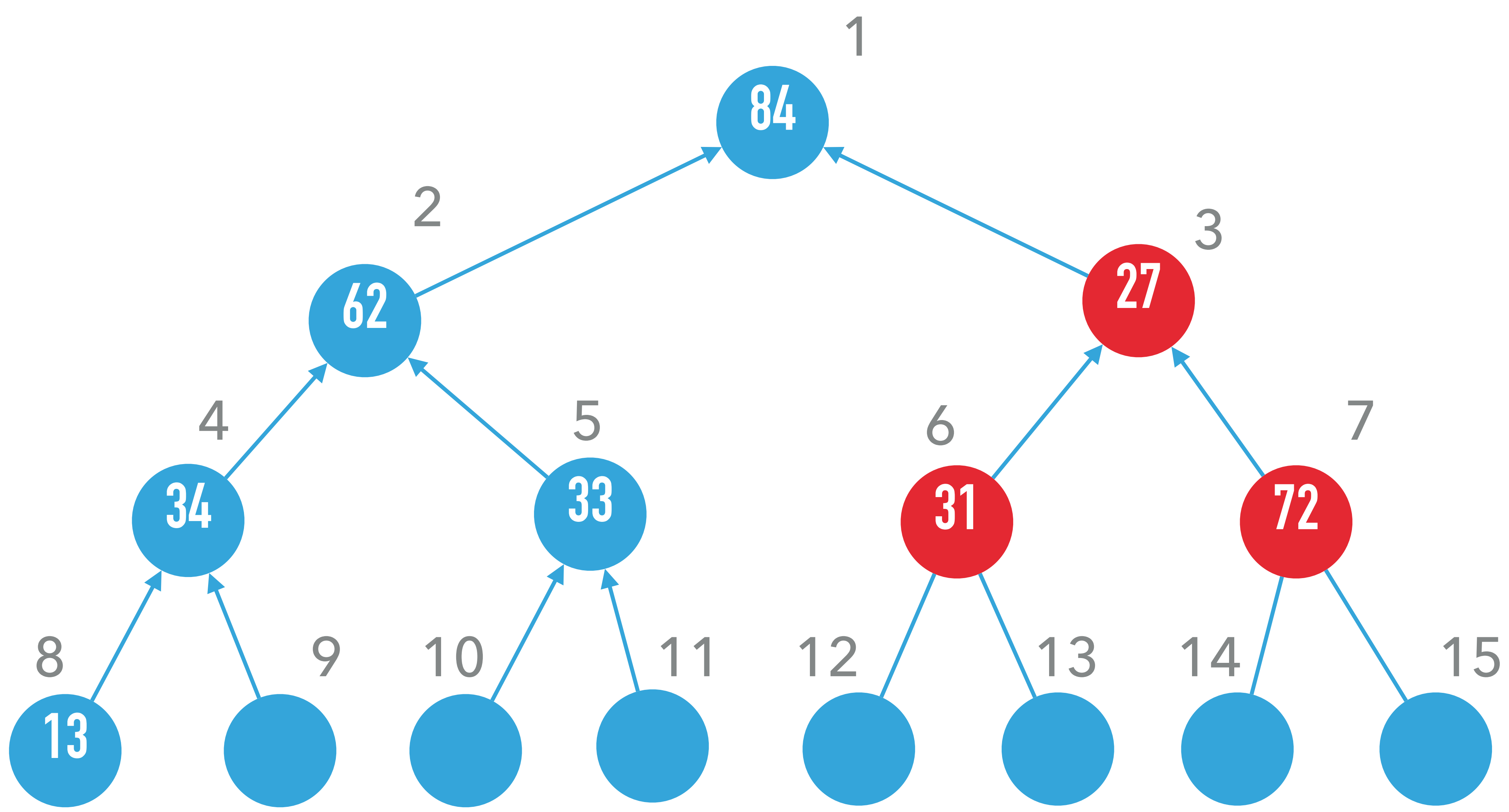
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	84	62	27	34	33	31	72	13							



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

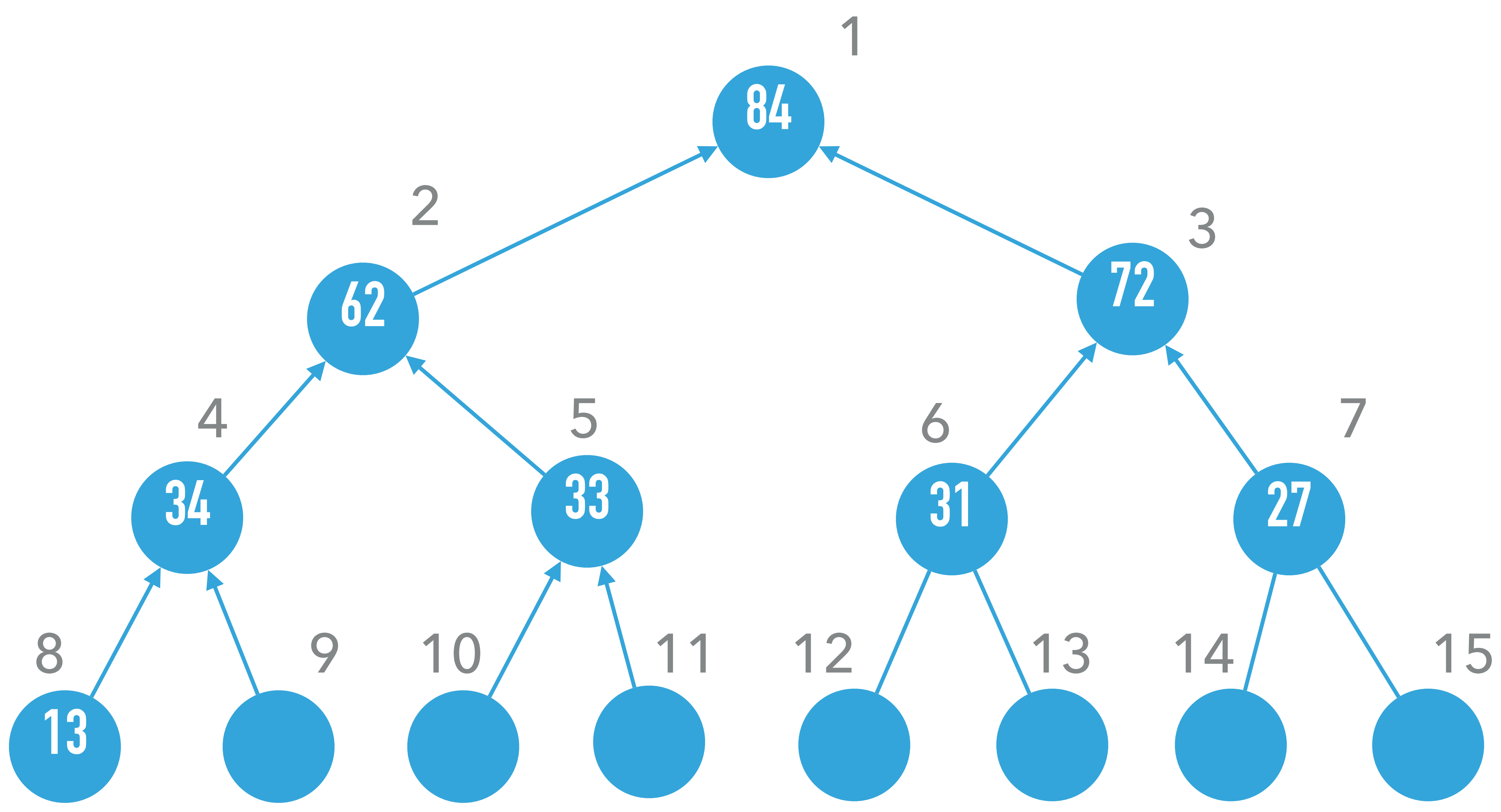
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	84	62	72	34	33	31	27	13							



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

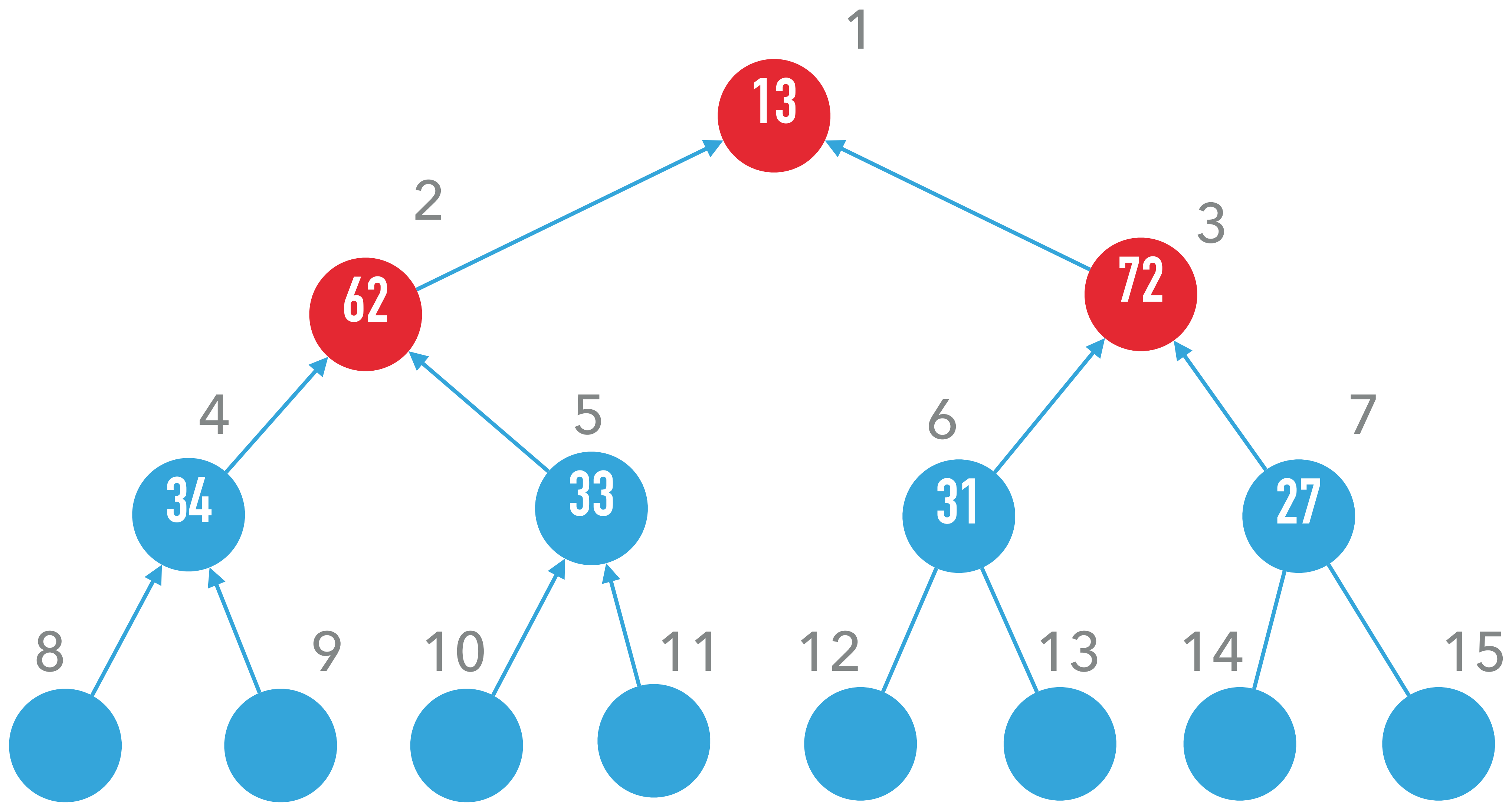
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	13	62	72	34	33	31	27								



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

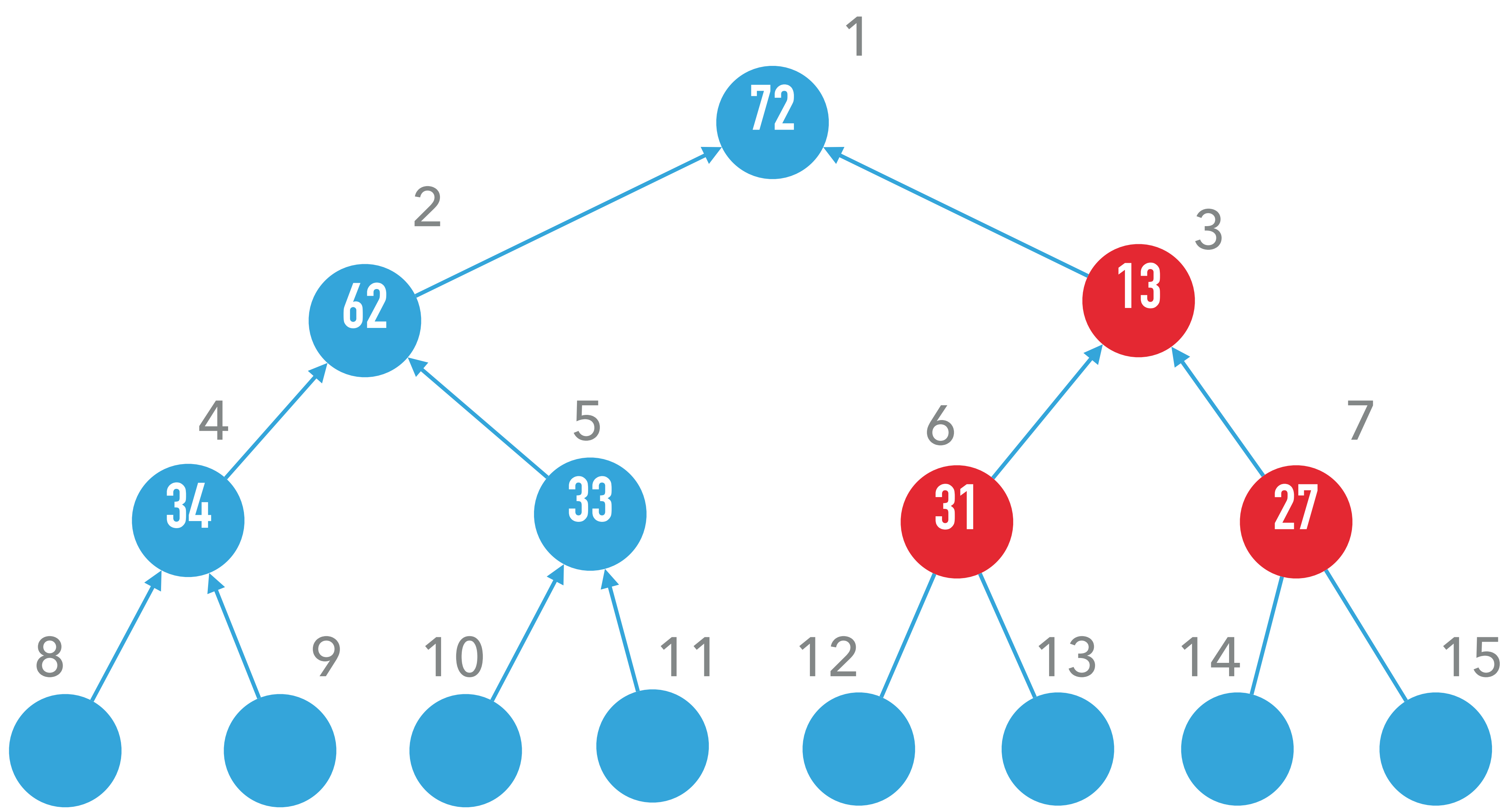
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	72	62	13	34	33	31	27								



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

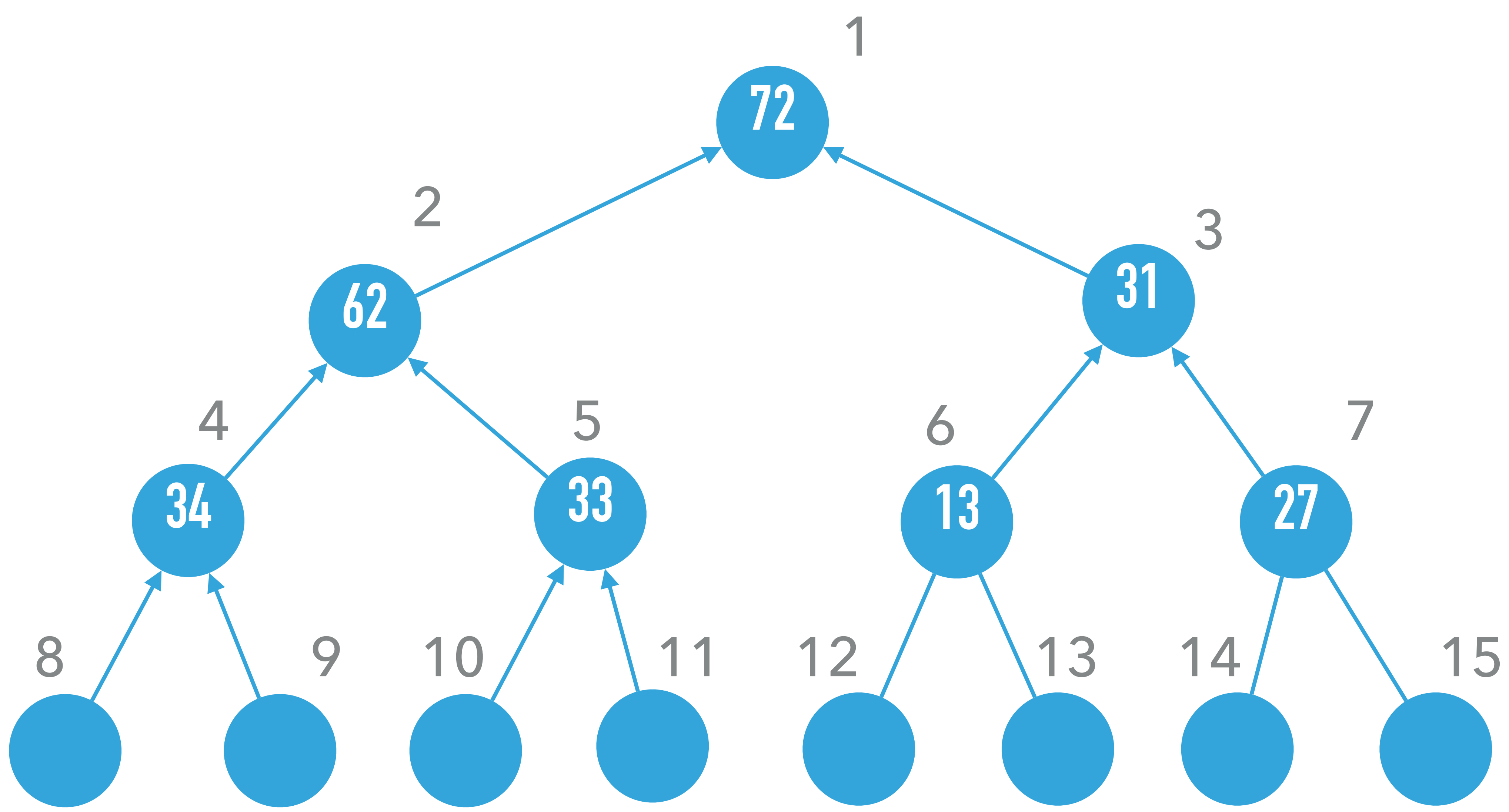
RETIRER MAX

RETIRER MAX

RETIRER MAX

Question 5.1.5 HEAPS, RETIRER LE MAX

Idx	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	72	62	31	34	33	13	27								



IDEE 2: COULER

Enfant de gauche de i : 2*i

Enfant de droite de i: 2*i+1

RETIRER MAX

RETIRER MAX

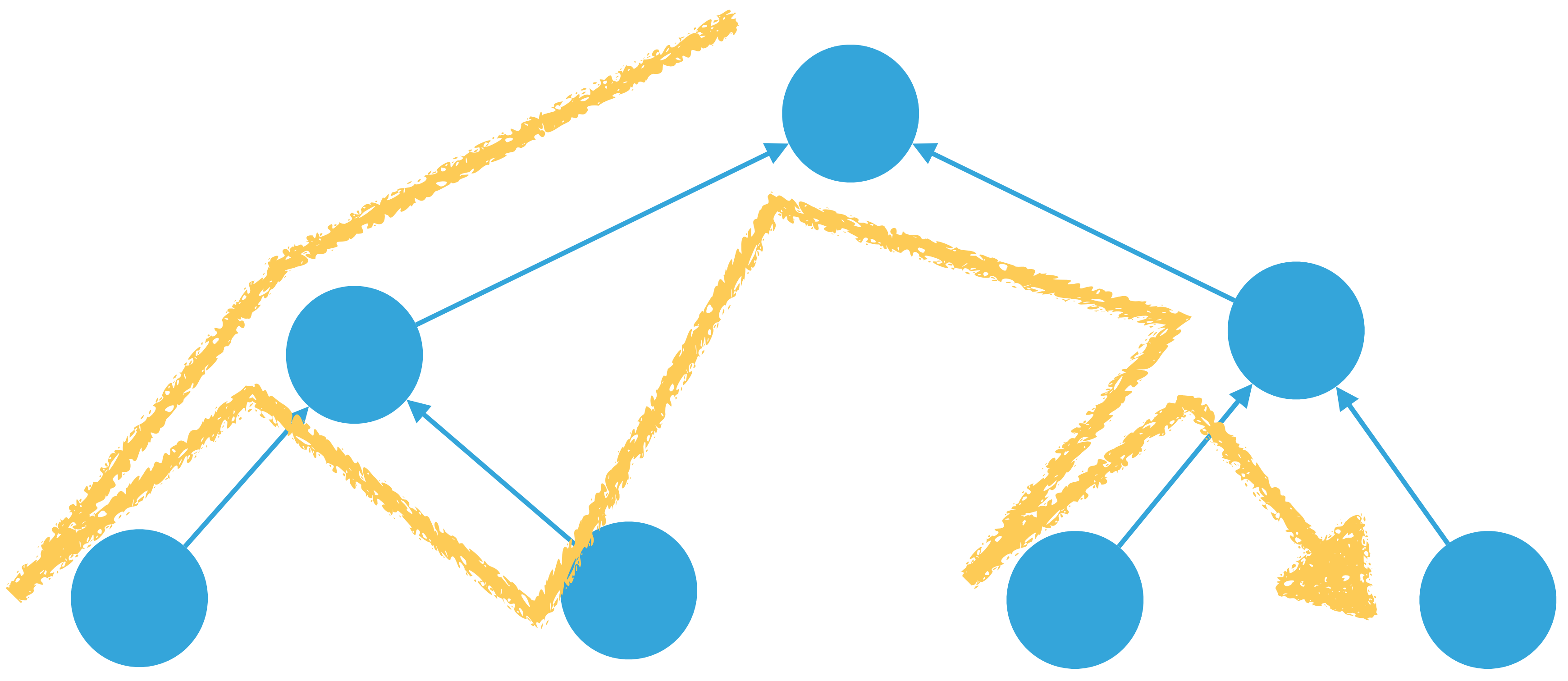
RETIRER MAX

Question 5.1.6 PQ: HEAP vs FILE

	Liste	Heap
Insert	O(n)	O(log n)
Pop max	O(1)	O(log n)

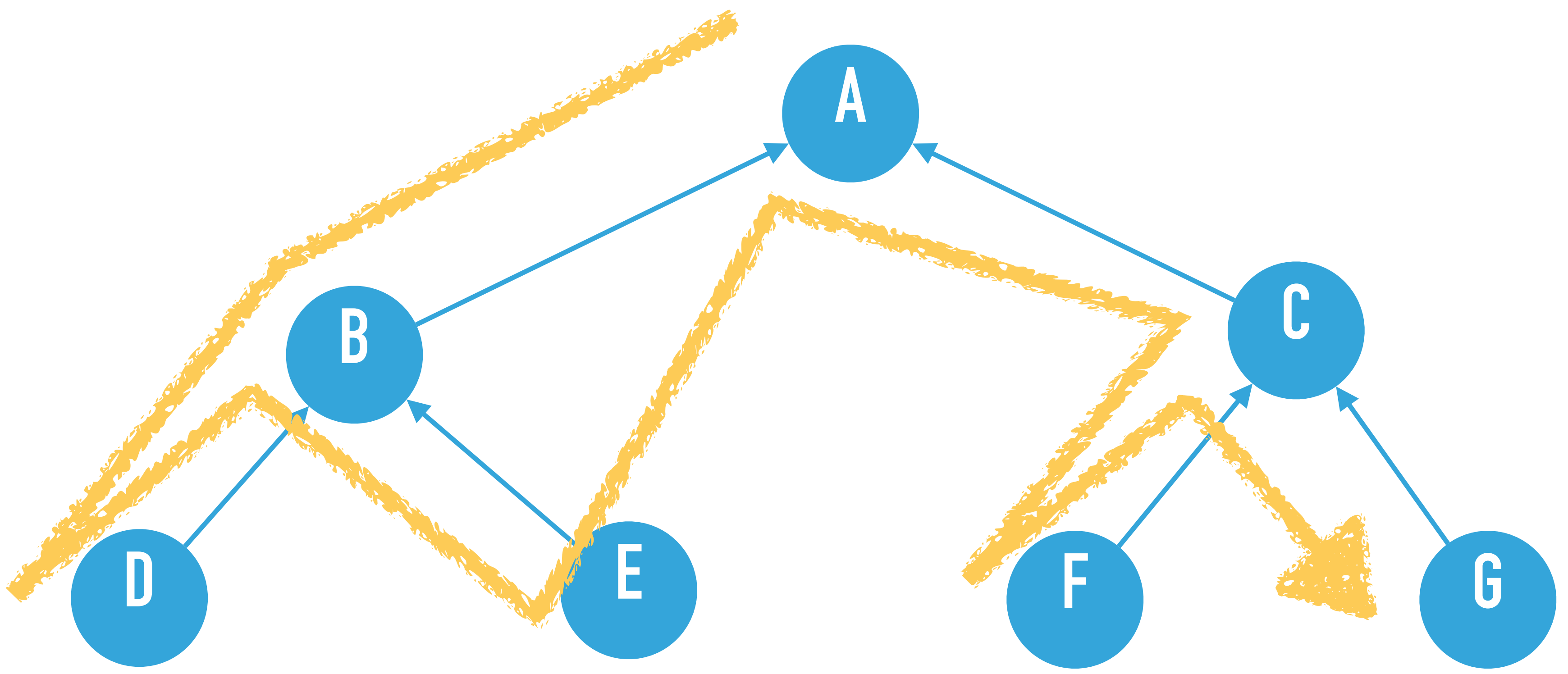
Question 5.1.7 Heap et ordre de traversée

Existe t'il un tas T mémorisant 7 éléments distincts tel qu'un parcours infixe du tas renvoie les éléments dans l'ordre (croissant ou décroissant) ? Et avec un parcours pre/postfixe?



Question 5.1.7 Heap et ordre de traversée

Existe t'il un tas T mémorisant 7 éléments distincts tel qu'un parcours infixe du tas renvoie les éléments dans l'ordre (croissant ou décroissant) ? Et avec un parcours pre/postfixe?



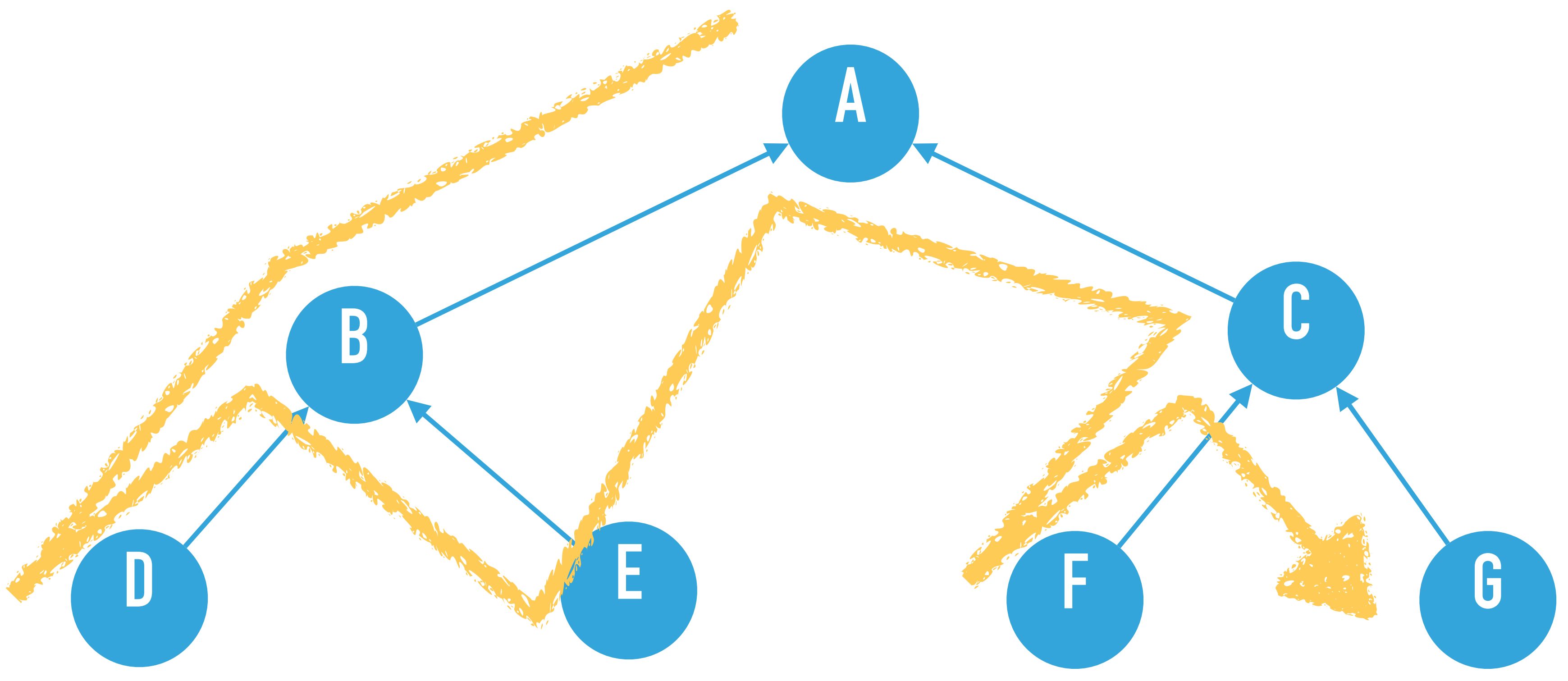
Parcours infixe: DBEAFCG

Parcours préfixe: ABDECFG

Parcours postfixe: DEBFGCA

Question 5.1.7 Heap et ordre de traversée

Existe t'il un tas T mémorisant 7 éléments distincts tel qu'un parcours infixe du tas renvoie les éléments dans l'ordre (croissant ou décroissant) ? Et avec un parcours pre/postfixe?



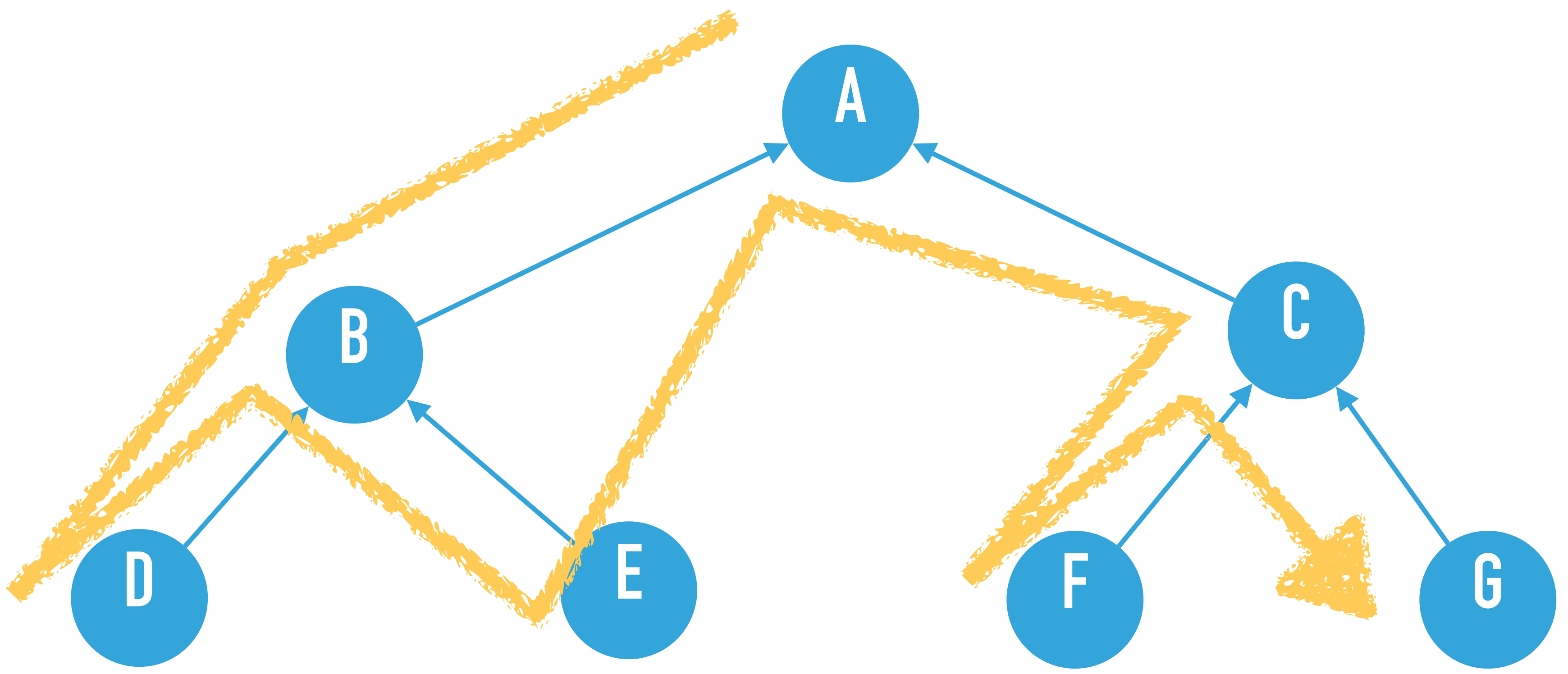
~~Parcours infixe: DBEAFCG~~

Parcours préfixe: ABDECFG

Parcours postfixe: DEBFGCA

Question 5.1.7 Heap et ordre de traversée

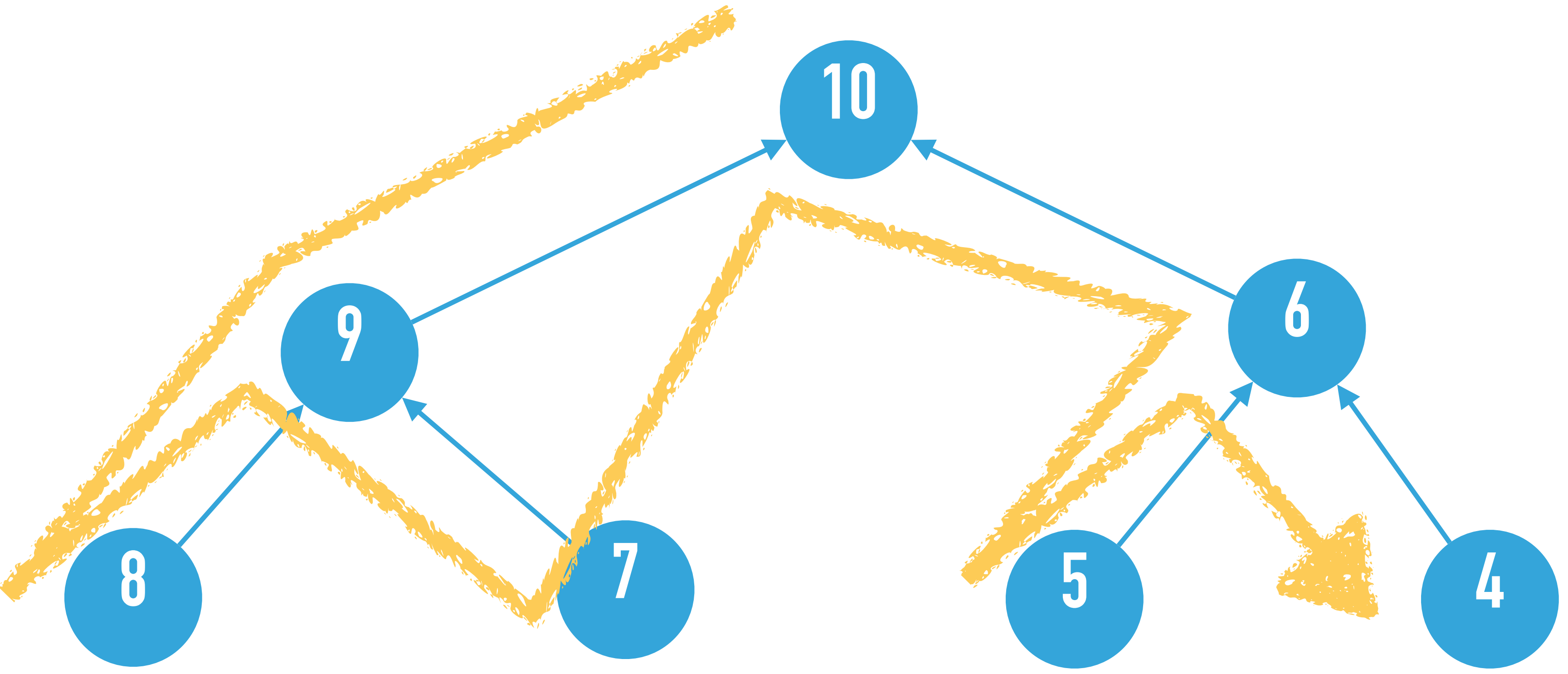
Existe t'il un tas T mémorisant 7 éléments distincts tel qu'un parcours infixe du tas renvoie les éléments dans l'ordre (croissant ou décroissant) ? Et avec un parcours pre/postfixe?

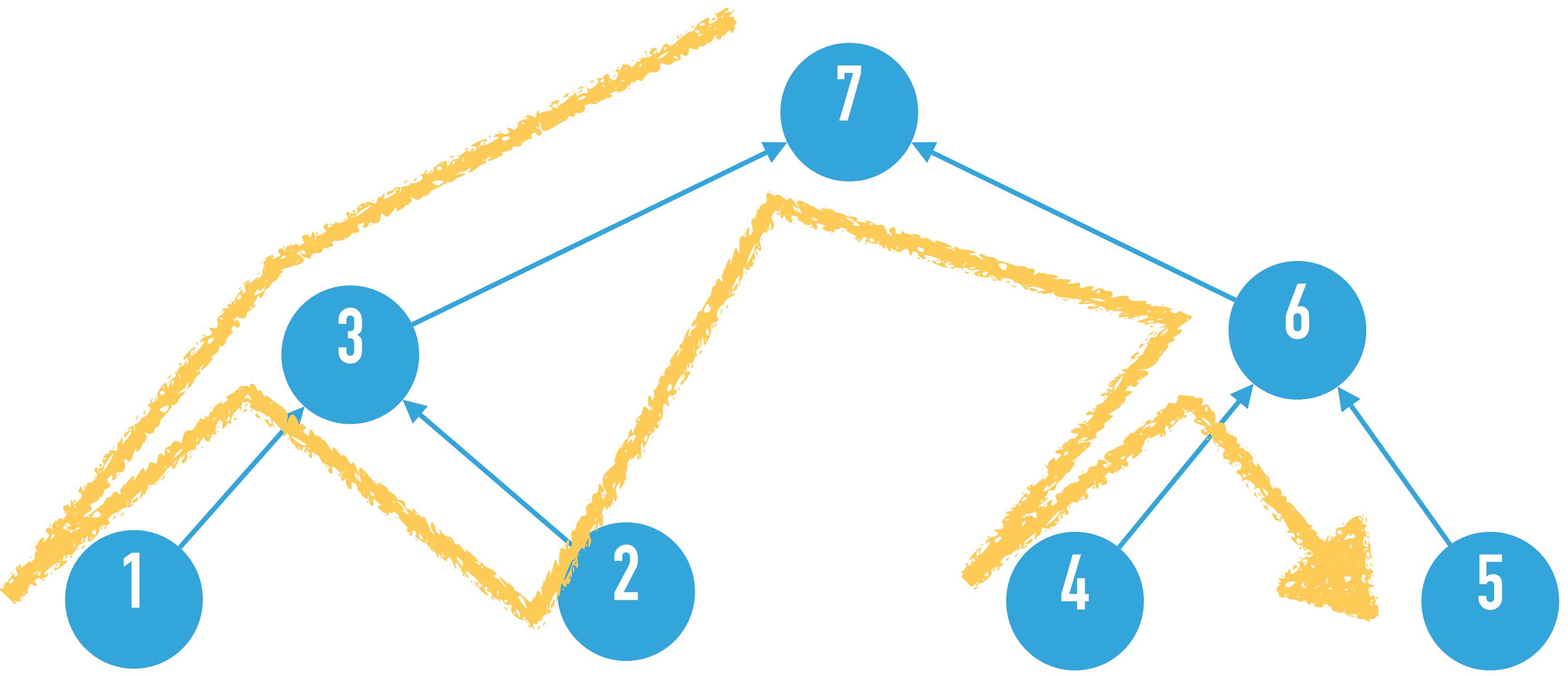


~~Parcours infixe: DBEAFCG~~

Parcours préfixe: ABDECFG

Parcours postfixe: DEBFGCA





Question 5.1.8 HEAPS et propriétés

- Dans le pire cas, l'insertion d'une clef dans une heap binaire est O(log n)
- Soit a[] tel que a[1] > a[2] > a[3] … > a[n]. Alors a[] est une heap binaire.
- Le tableau d'une heap max est toujours trié dans l'ordre décroissant
- Etant donné une heap binaire de N clefs distinctes, supprimer le max et le remettre laisse le tableau inchangé.

Question 5.1.8 HEAPS et propriétés

- Dans le pire cas, l'insertion d'une clef dans une heap binaire est O(log n)
- Soit a[] tel que a[0] > a[1] > a[2] > a[3] … > a[n]. Alors a[] est une heap binaire.
- Le tableau d'une heap est toujours trié dans l'ordre décroissant
- Etant donné une heap binaire de N clefs distinctes, supprimer le max et le remettre laisse le tableau inchangé.

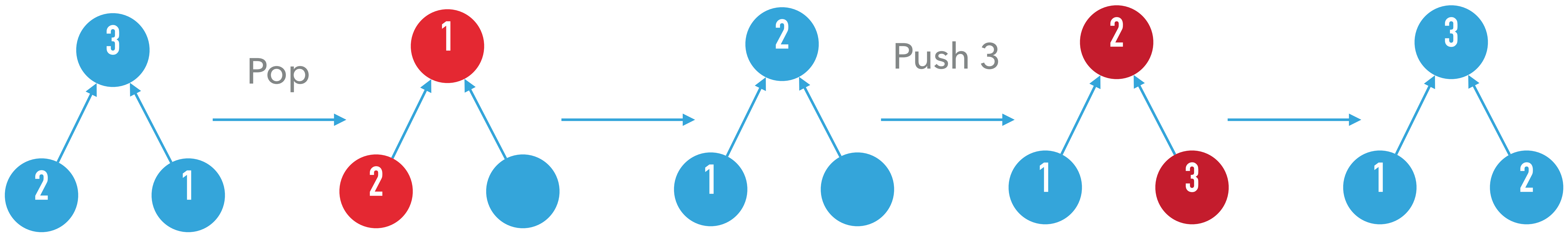
Question 5.1.8 HEAPS et propriétés

- Dans le pire cas, l'insertion d'une clef dans une heap binaire est O(log n) (and not ~O(log n) because best -case in O(1))

- Soit a[] tel que a[0] > a[1] > a[2] > a[3] … > a[n]. Alors a[] est une heap binaire.

- Le tableau d'une heap est toujours trié dans l'ordre décroissant

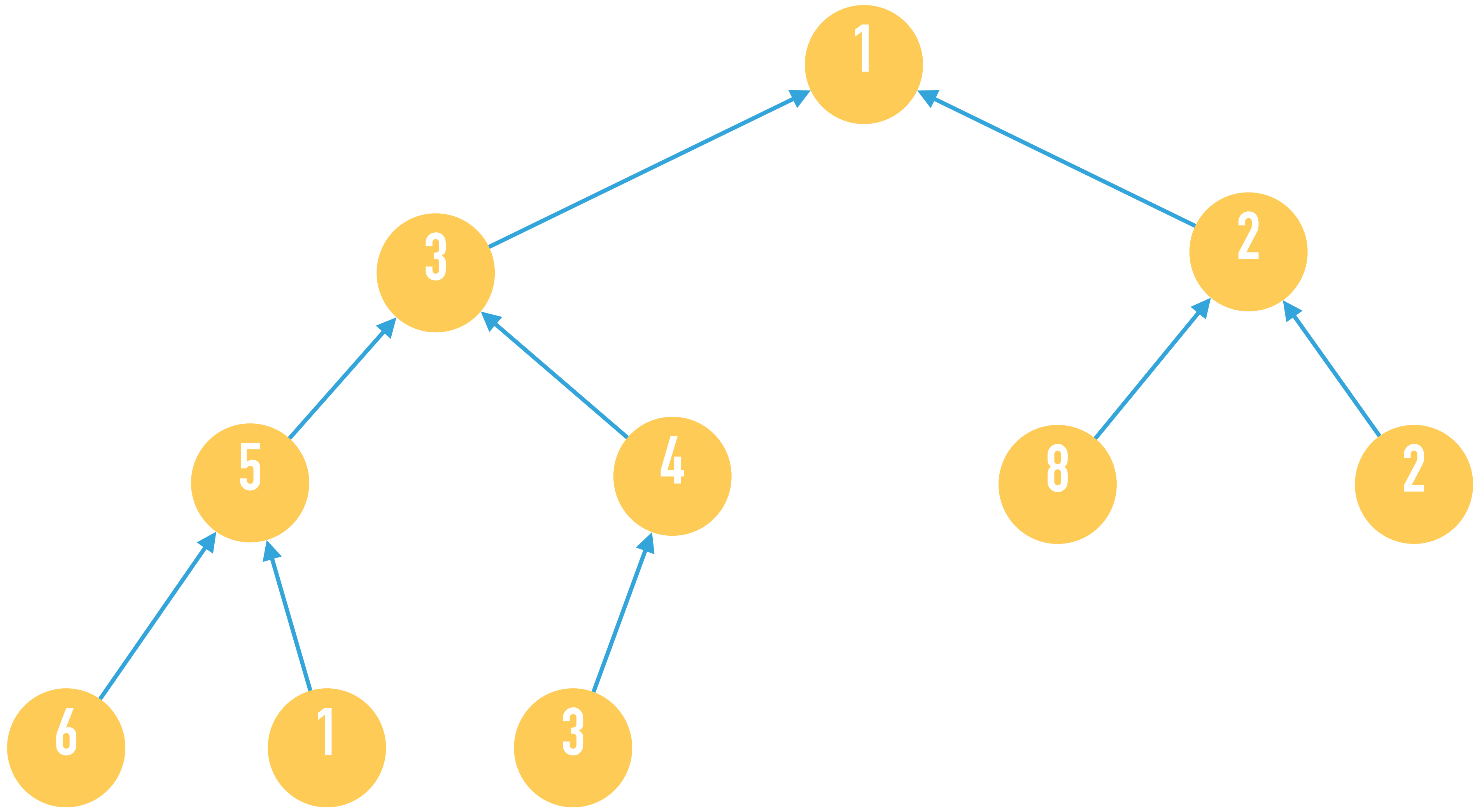
- Etant donné une heap binaire de N clefs distinctes, supprimer le max et le remettre laisse le tableau inchangé.



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire sink

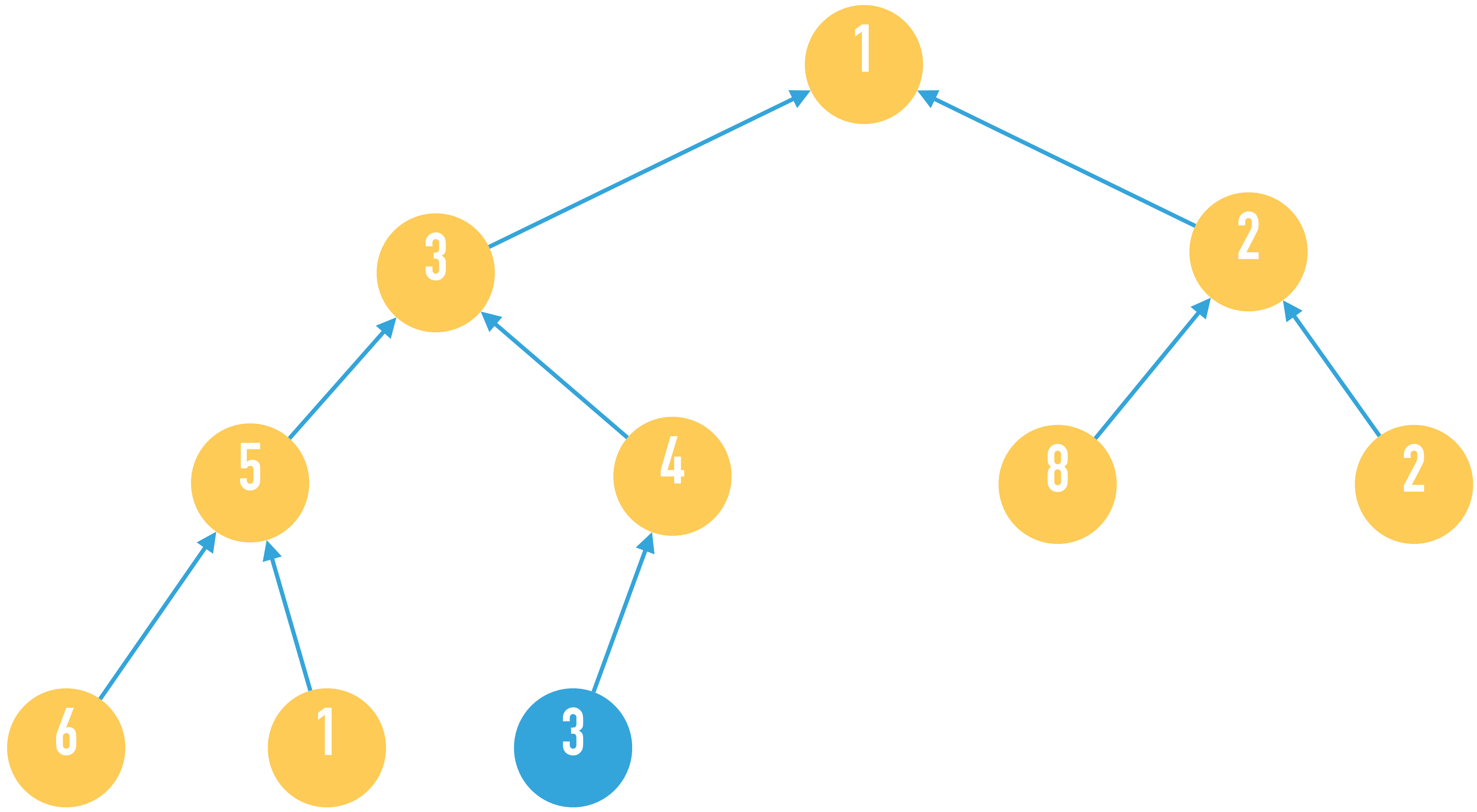
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	5	4	8	2	6	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

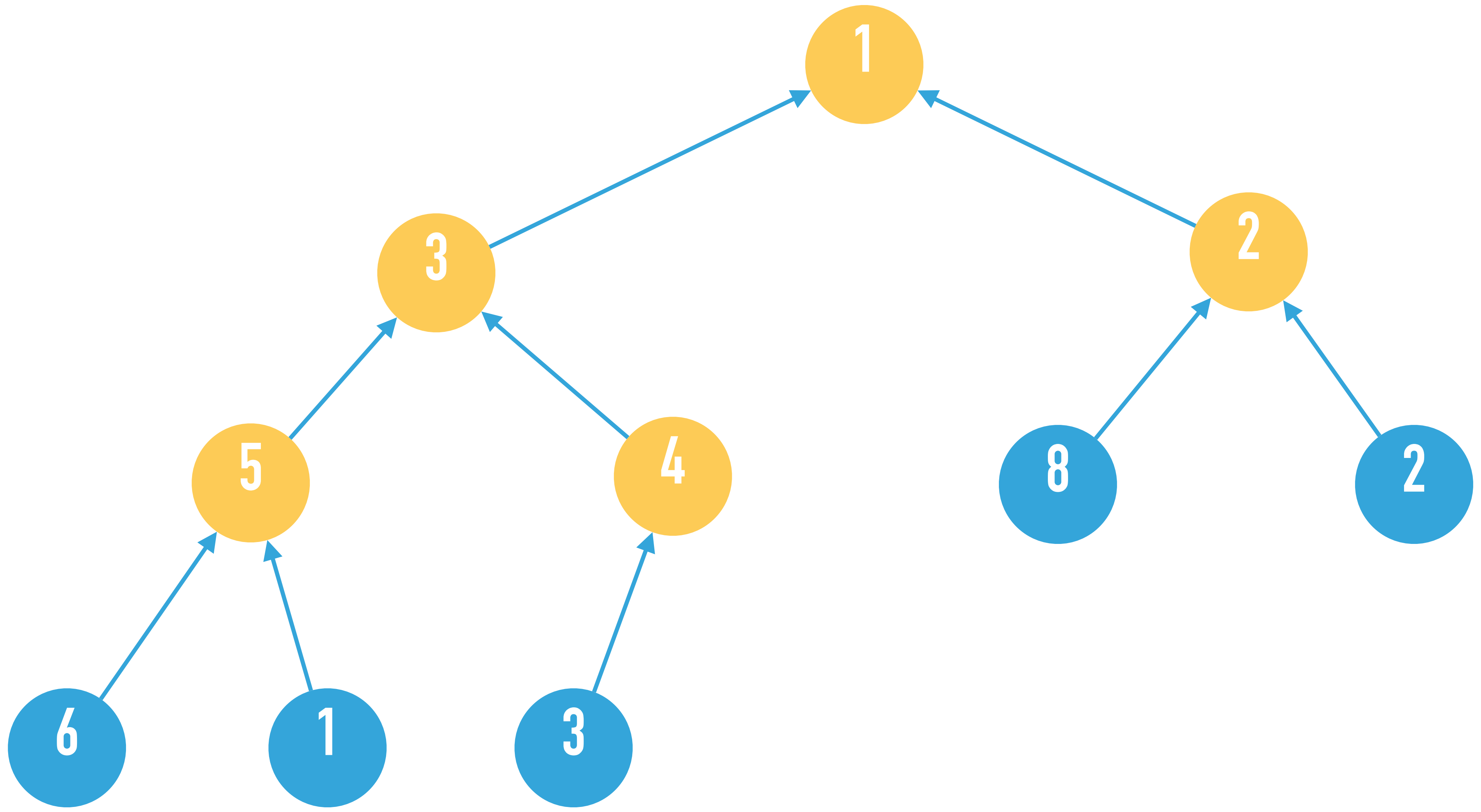
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	5	4	8	2	6	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

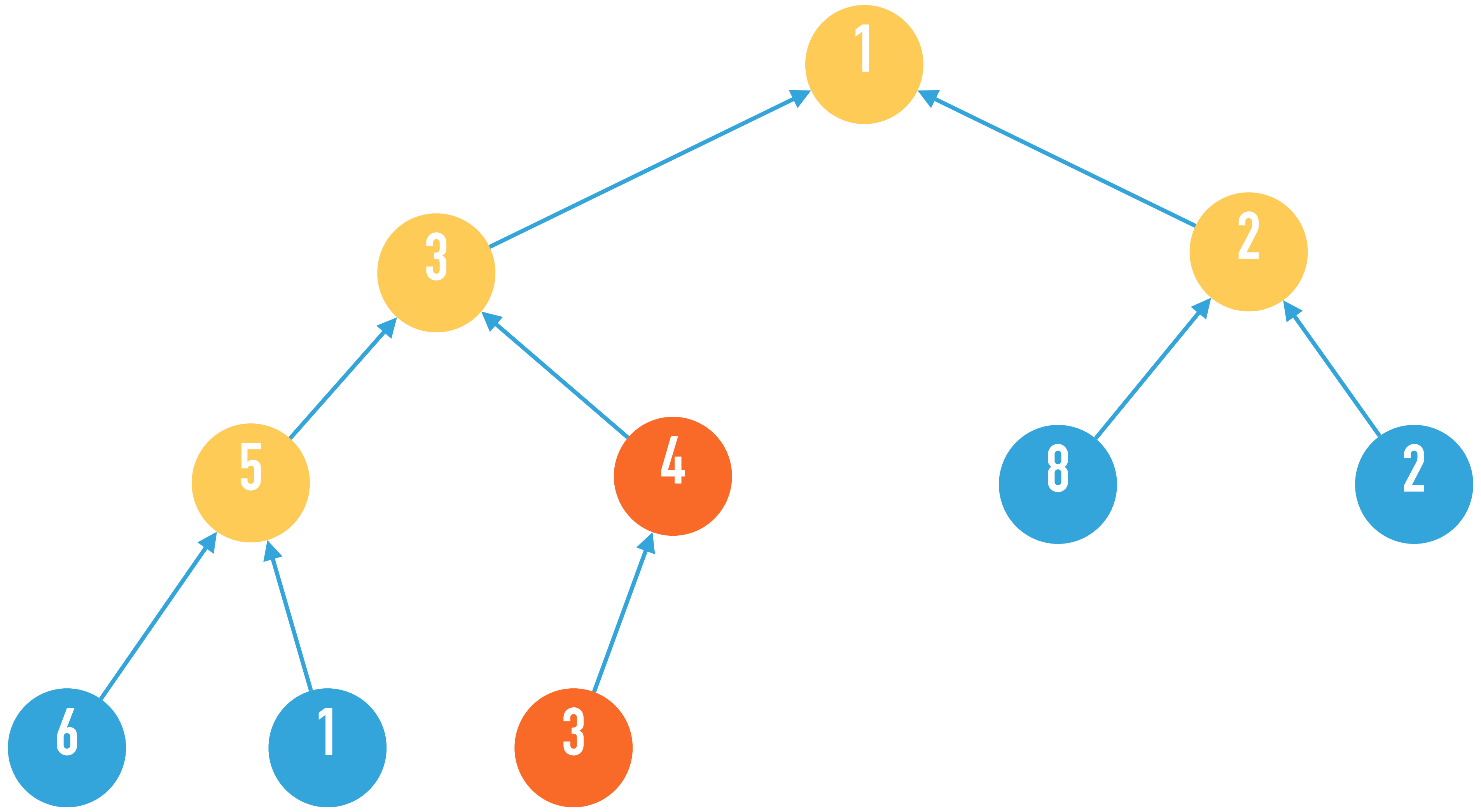
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	5	4	8	2	6	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

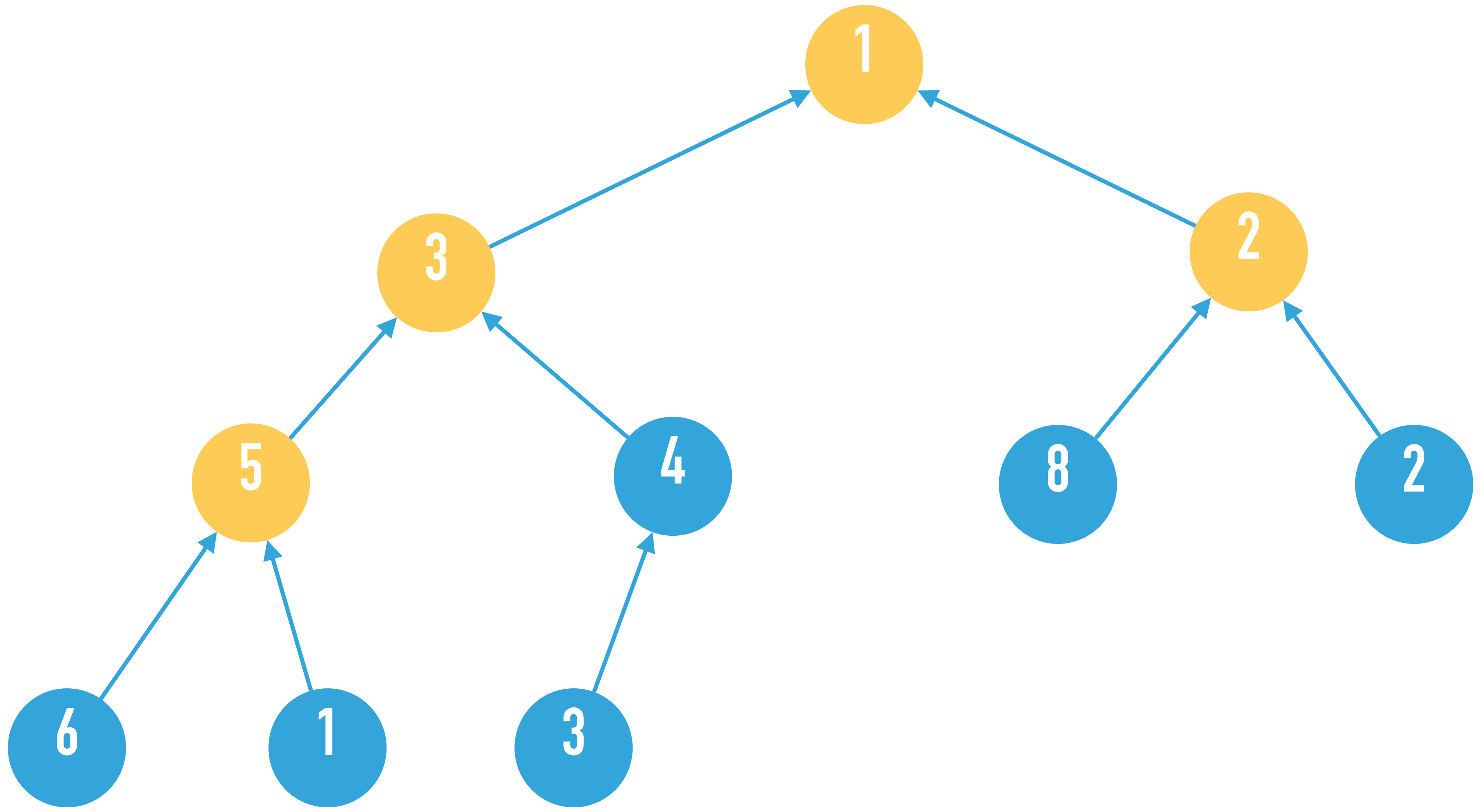
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	5	4	8	2	6	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

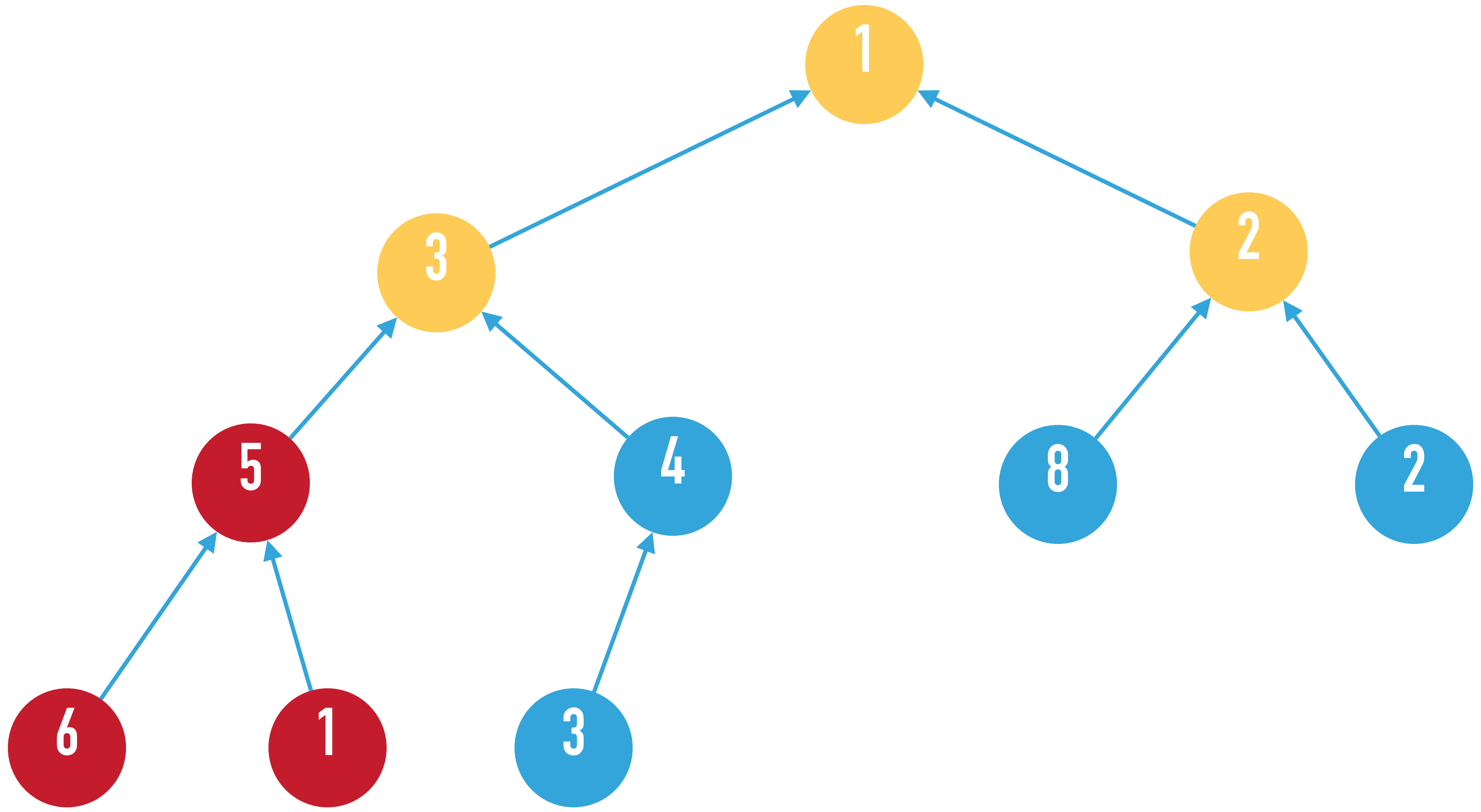
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	5	4	8	2	6	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

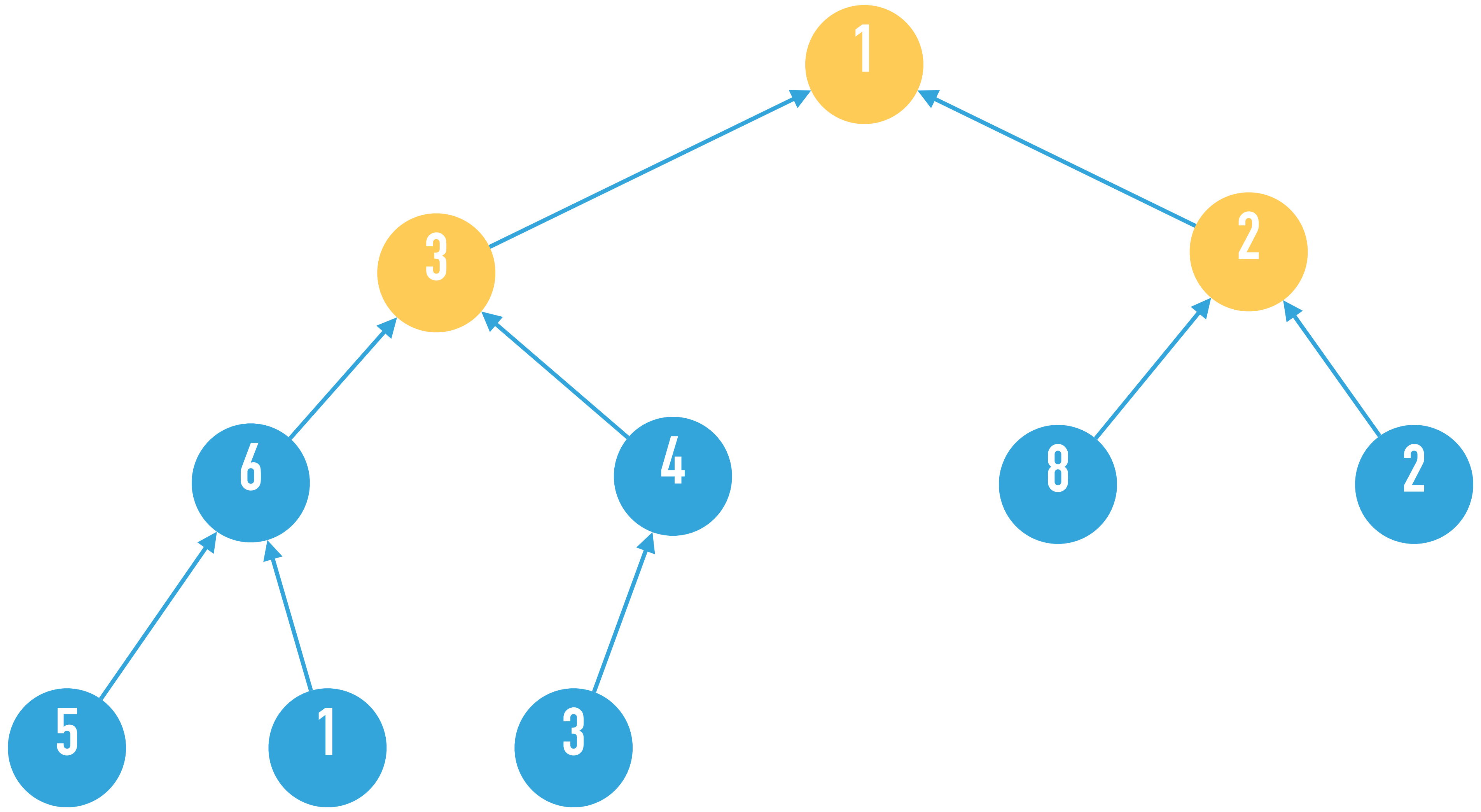
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	5	4	8	2	6	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

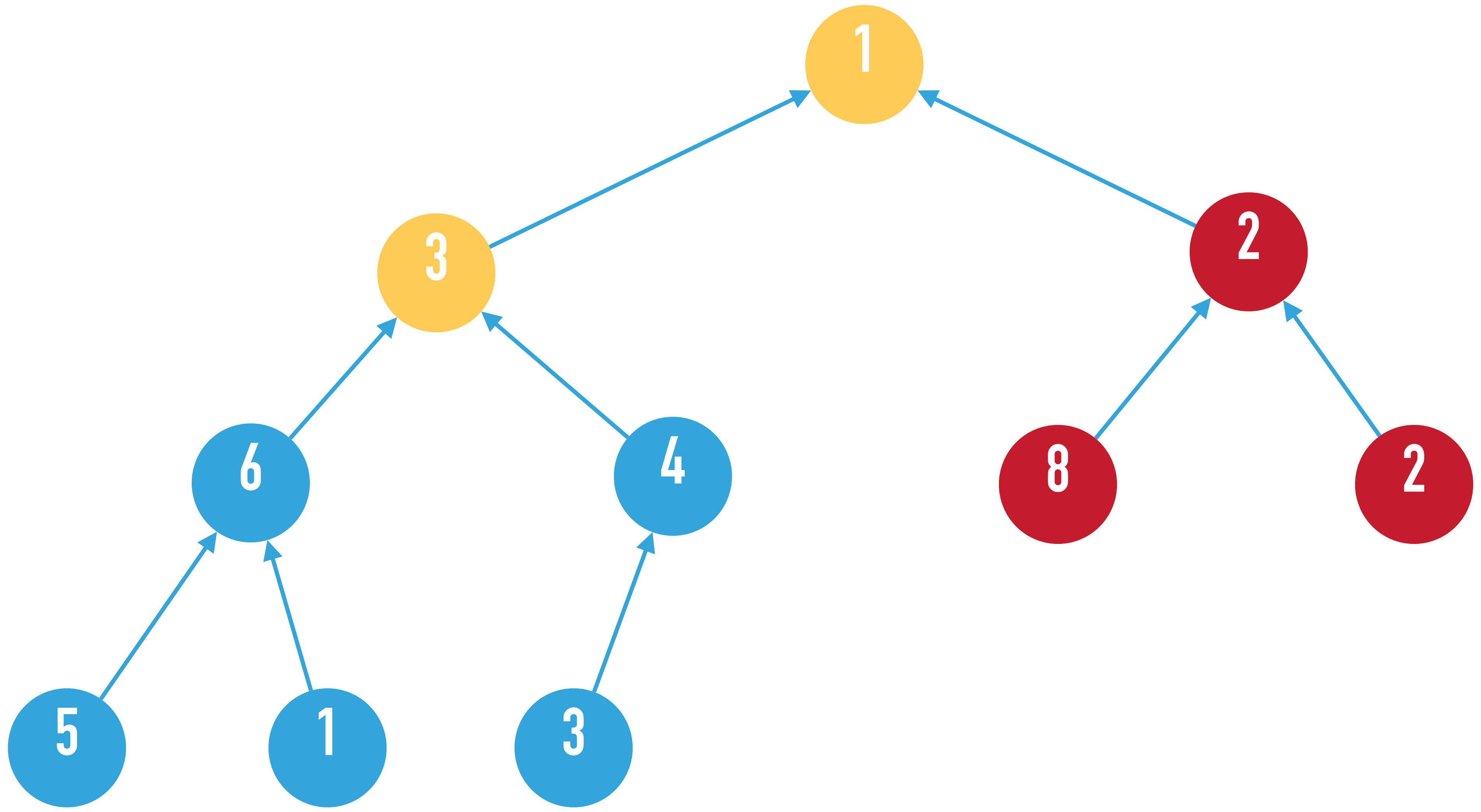
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	6	4	8	2	5	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

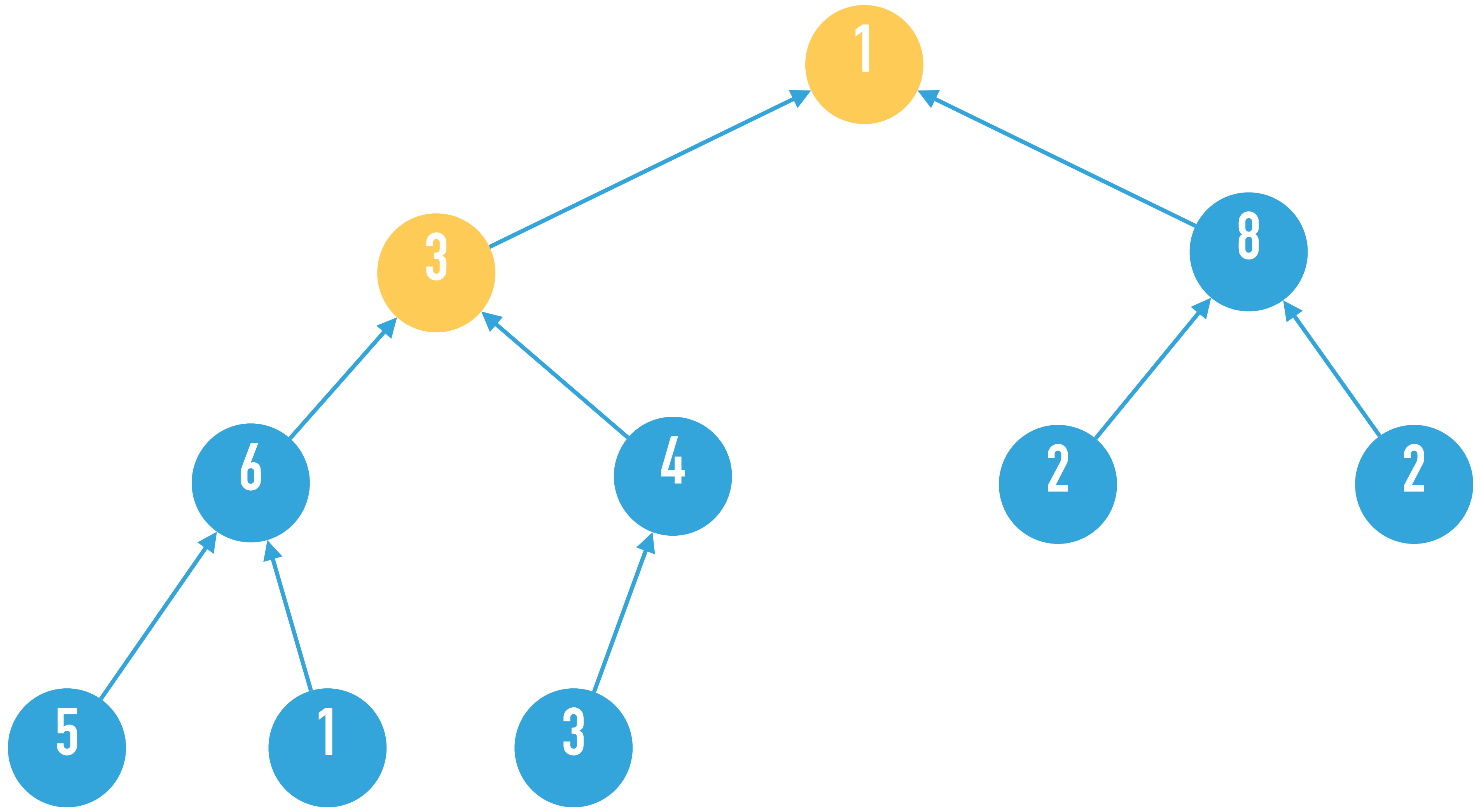
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	2	6	4	8	2	5	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

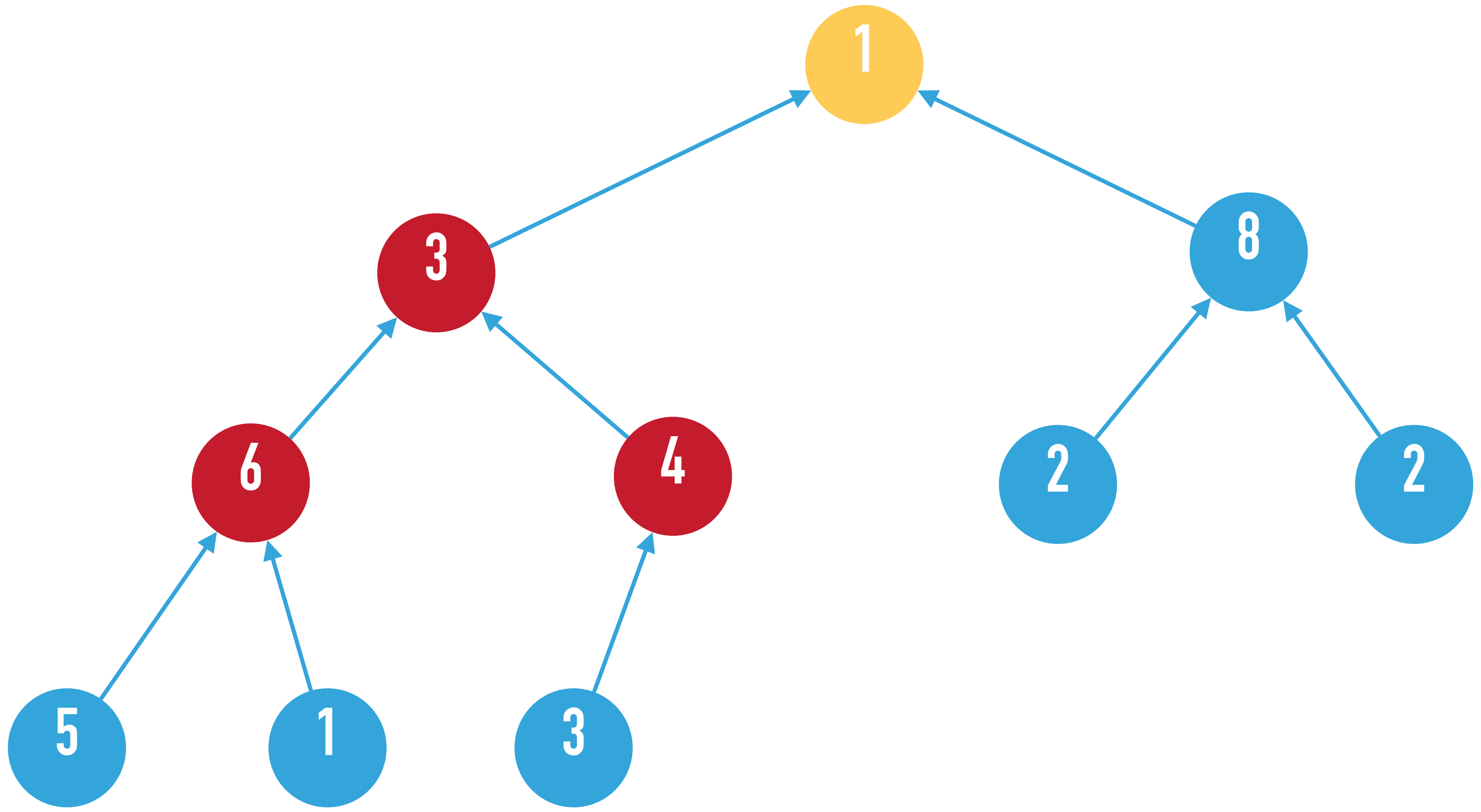
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	8	6	4	2	2	5	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

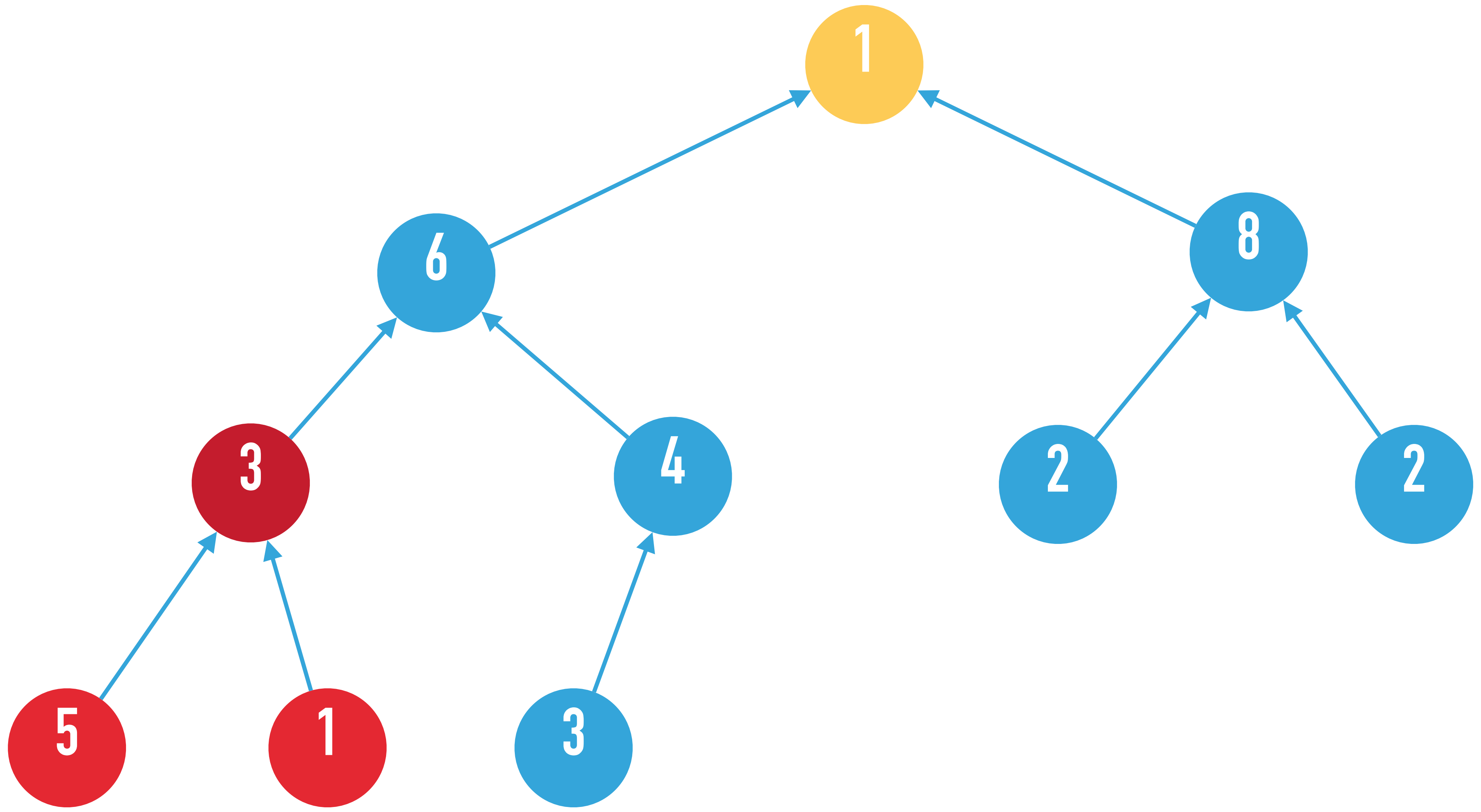
Idx	1	2	3	4	5	6	7	8	9	10
V	1	3	8	6	4	2	2	5	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

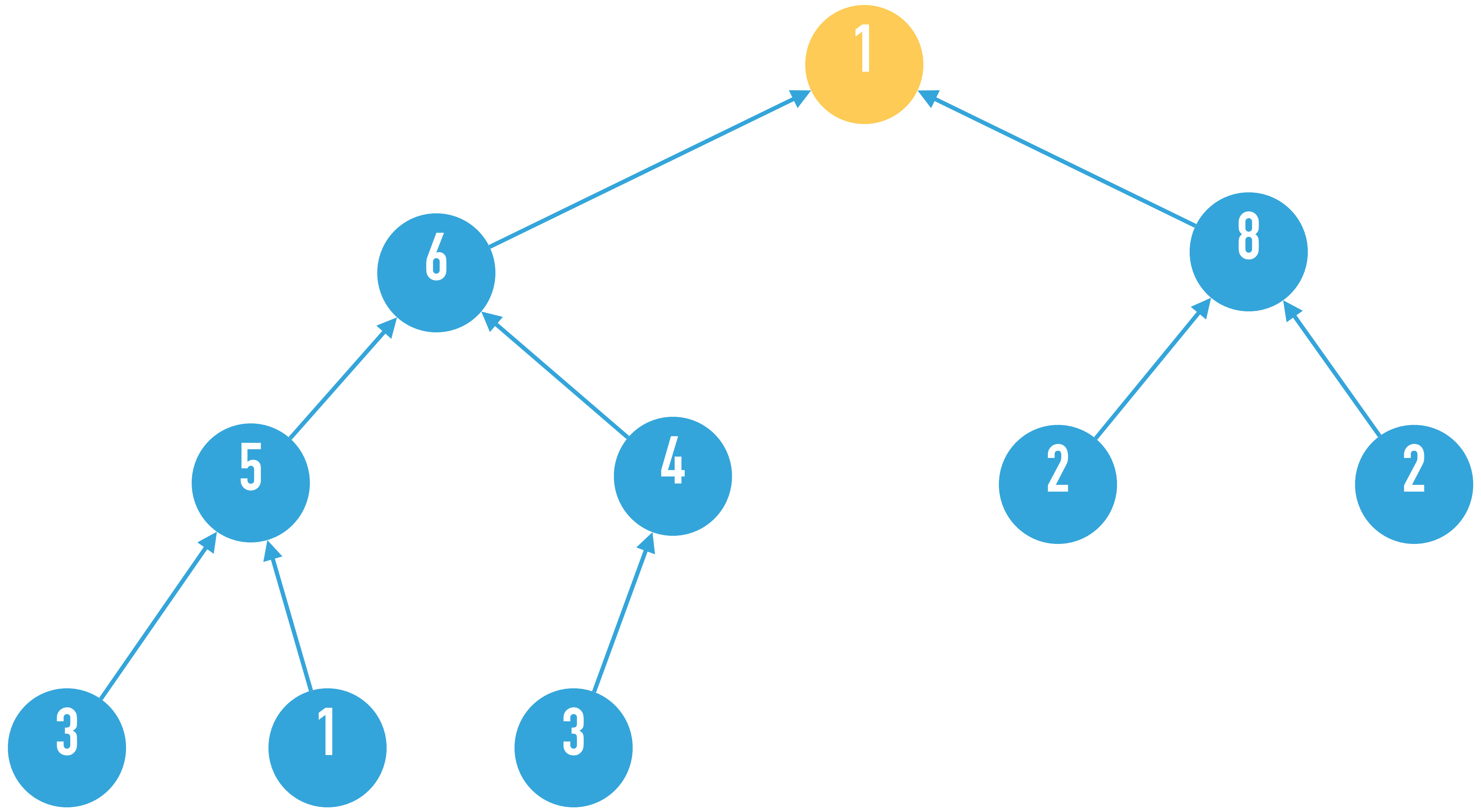
Idx	1	2	3	4	5	6	7	8	9	10
V	1	6	8	3	4	2	2	5	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

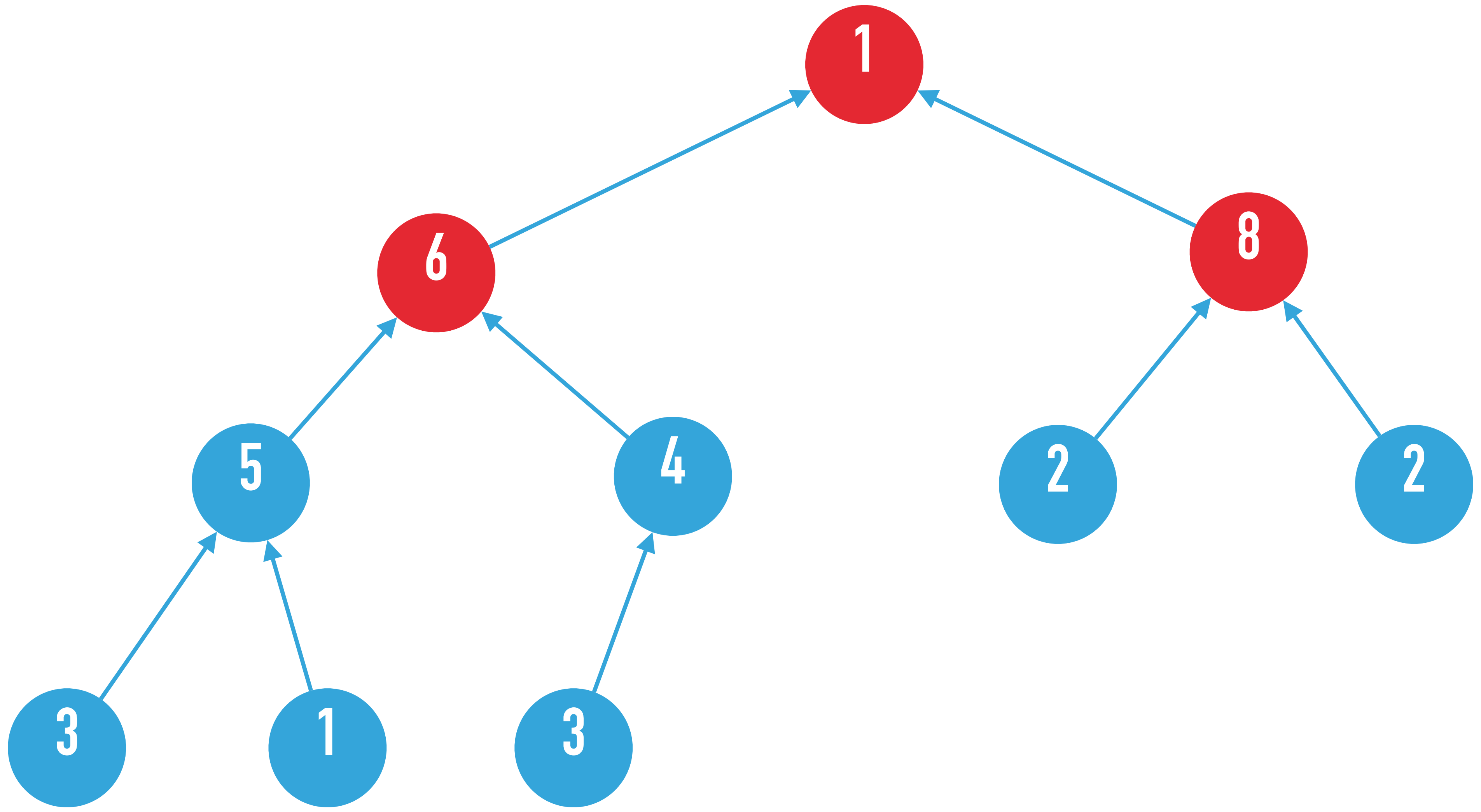
Idx	1	2	3	4	5	6	7	8	9	10
V	1	6	8	5	4	2	2	3	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

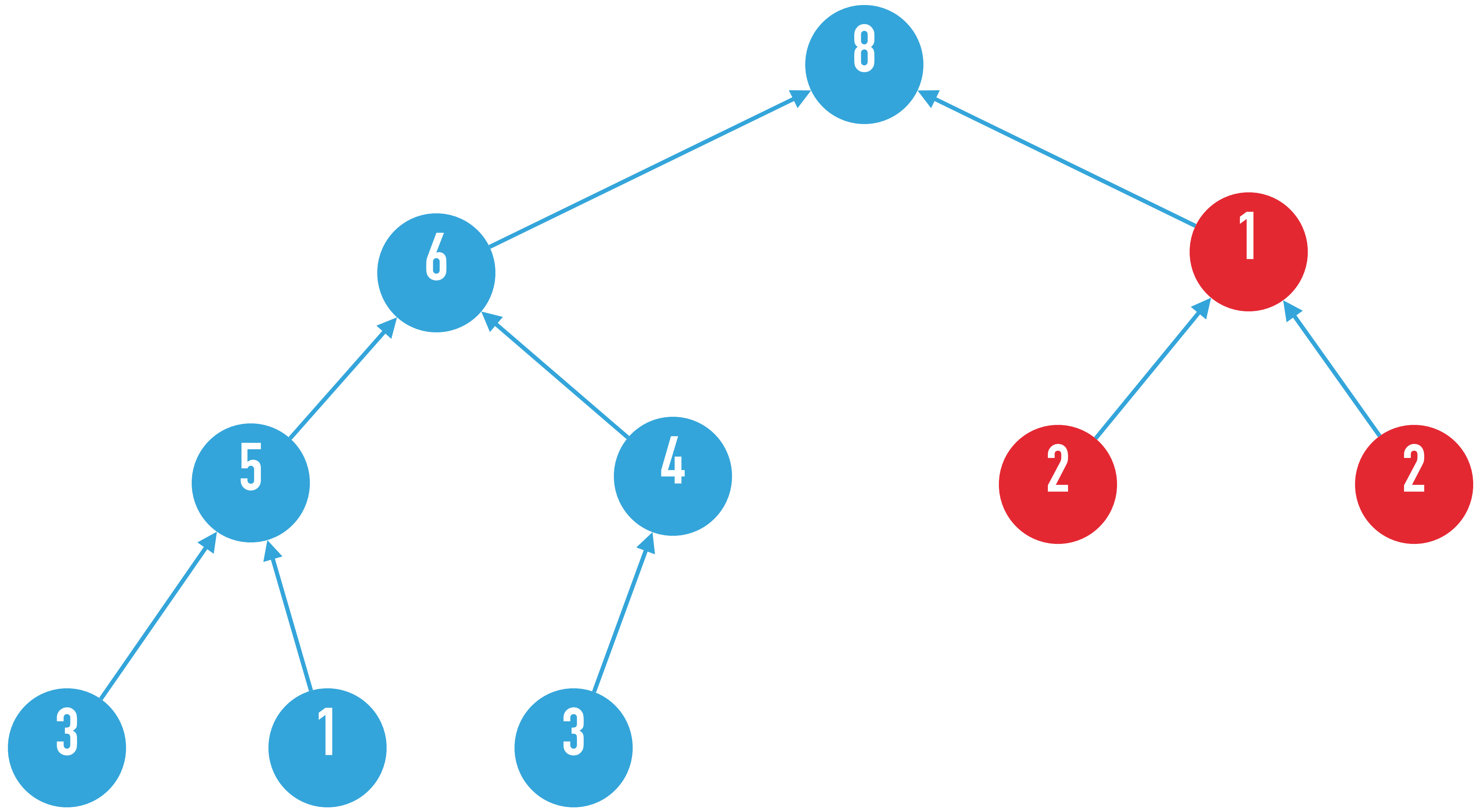
Idx	1	2	3	4	5	6	7	8	9	10
V	1	6	8	5	4	2	2	3	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

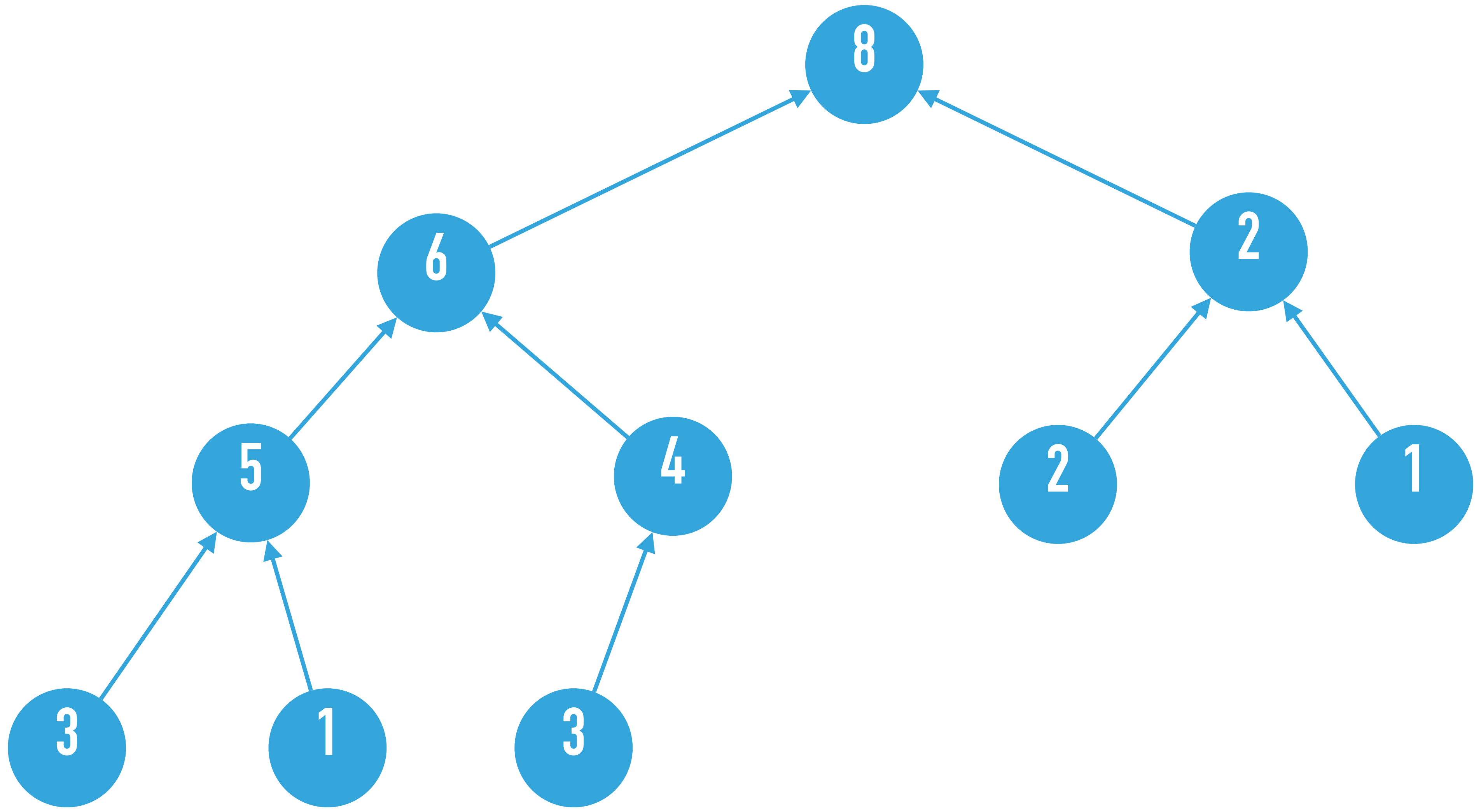
Idx	1	2	3	4	5	6	7	8	9	10
V	8	6	1	5	4	2	2	3	1	3



Question 5.1.9 HEAPIFY

On va considérer chaque noeud comme une racine d'un heap, tour à tour, et faire , en partant de la droite.

Idx	1	2	3	4	5	6	7	8	9	10
V	8	6	2	5	4	2	1	3	1	3



Question 5.1.9 HEAPIFY

Prouvez qu'on fait cette opération en O(n)

Pour n=2^m - 1, on a:

- 2^{m-1} tableaux de taille 1 à heapifier. Coût $2^{m-1} \cdot \log(1) = 0$
- 2^{m-2} tableaux de taille 3. $2^{m-2} \cdot \log(3) \sim 2 \cdot 2^{m-2}$
- 2^{m-3} tableaux de taille 7. $2^{m-3} \cdot \log(7) \sim 3 \cdot 2^{m-3}$
- ....

$$\sum_{i=2}^{m} i \cdot 2^{m-i} = \sum_{i=2}^{m} i \cdot \frac{n}{2^i} = n \sum_{i=2}^{m} i \cdot (\frac{1}{2})^i$$

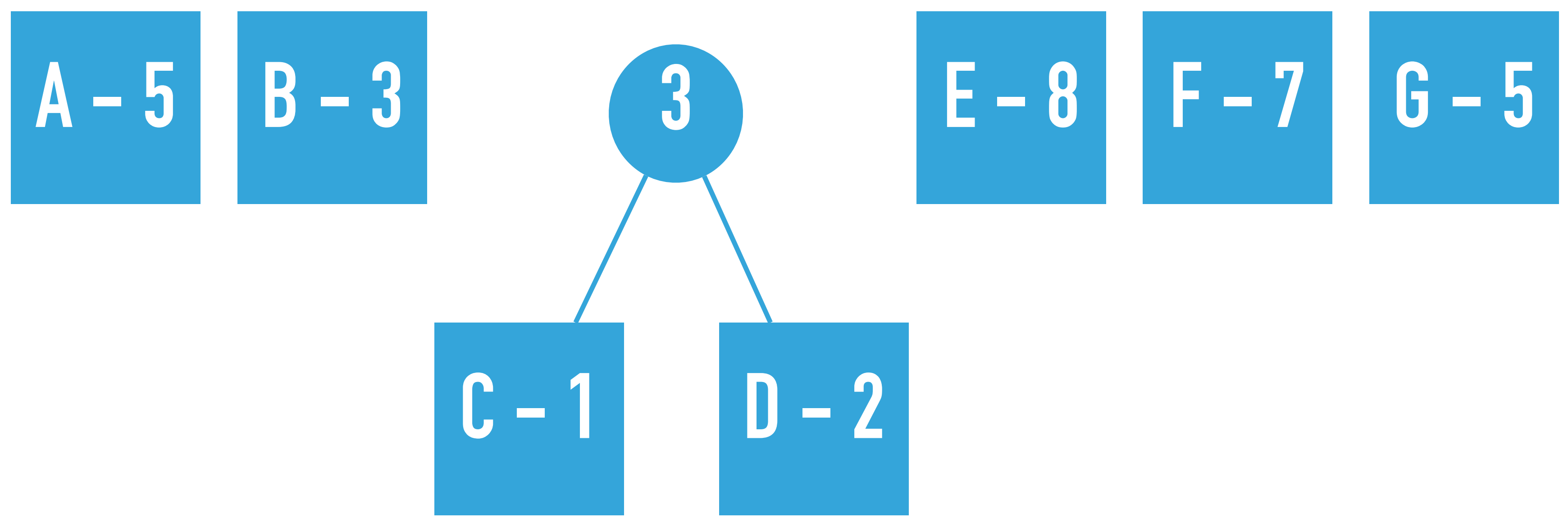
Or on a que $\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$ si $|x| < 1$

D'où $n \sum_{i=2}^{m} i \cdot (\frac{1}{2})^i \leq 2n$

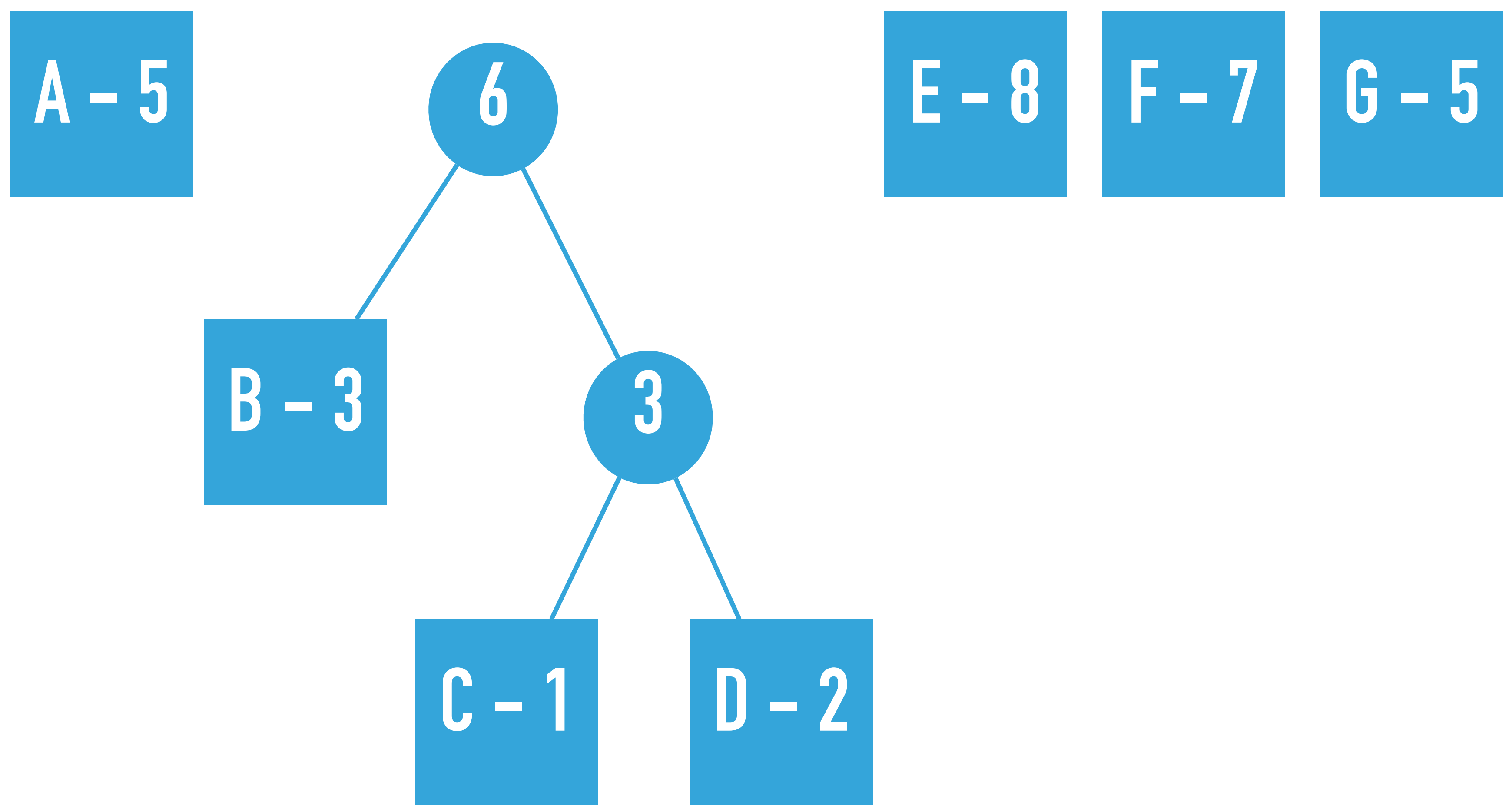
Question 5.1.10 Huffman

A - 5 | B - 3 | C - 1 | D - 2 | E - 8 | F - 7 | G - 5

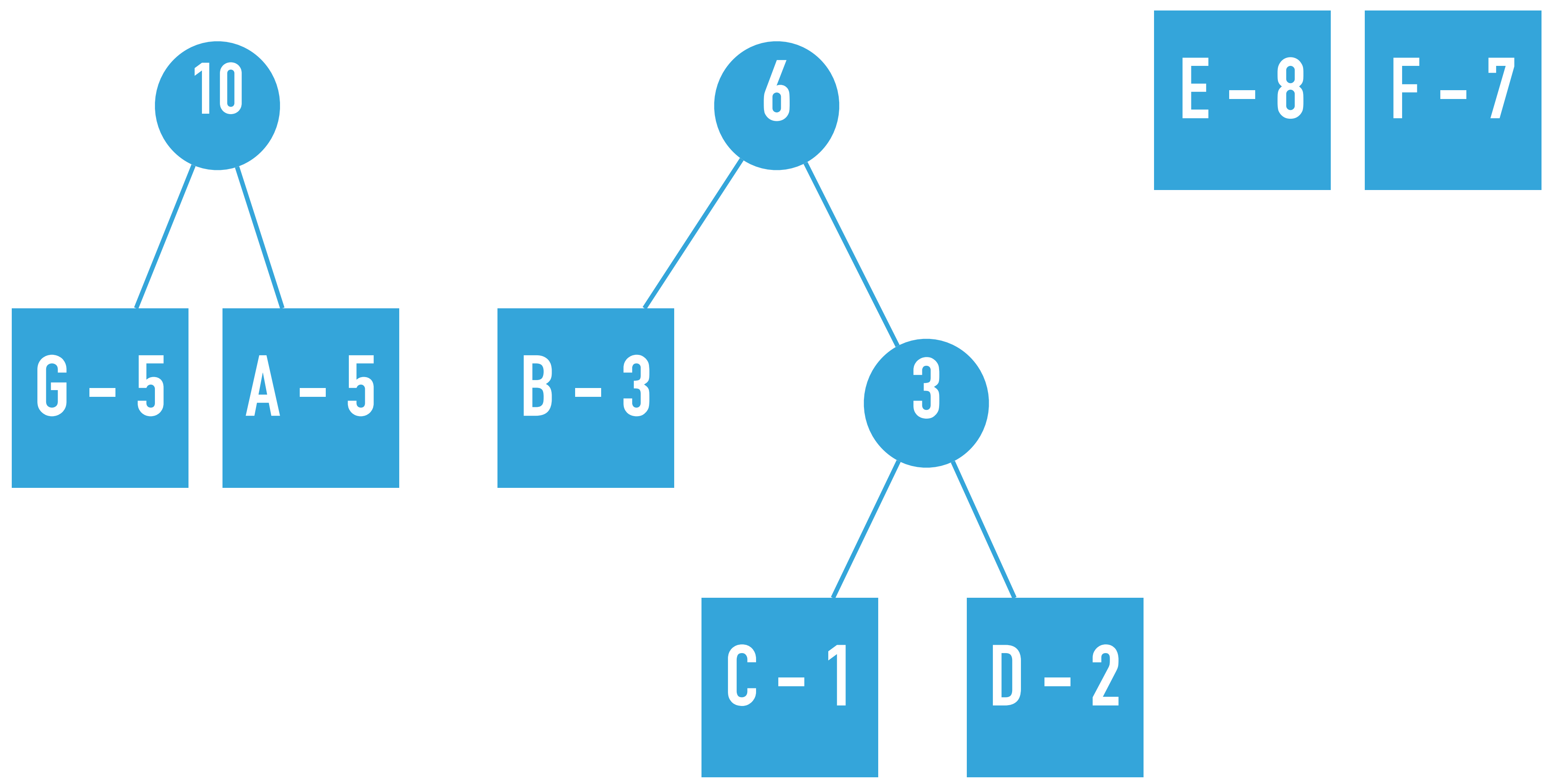
Question 5.1.10 Huffman



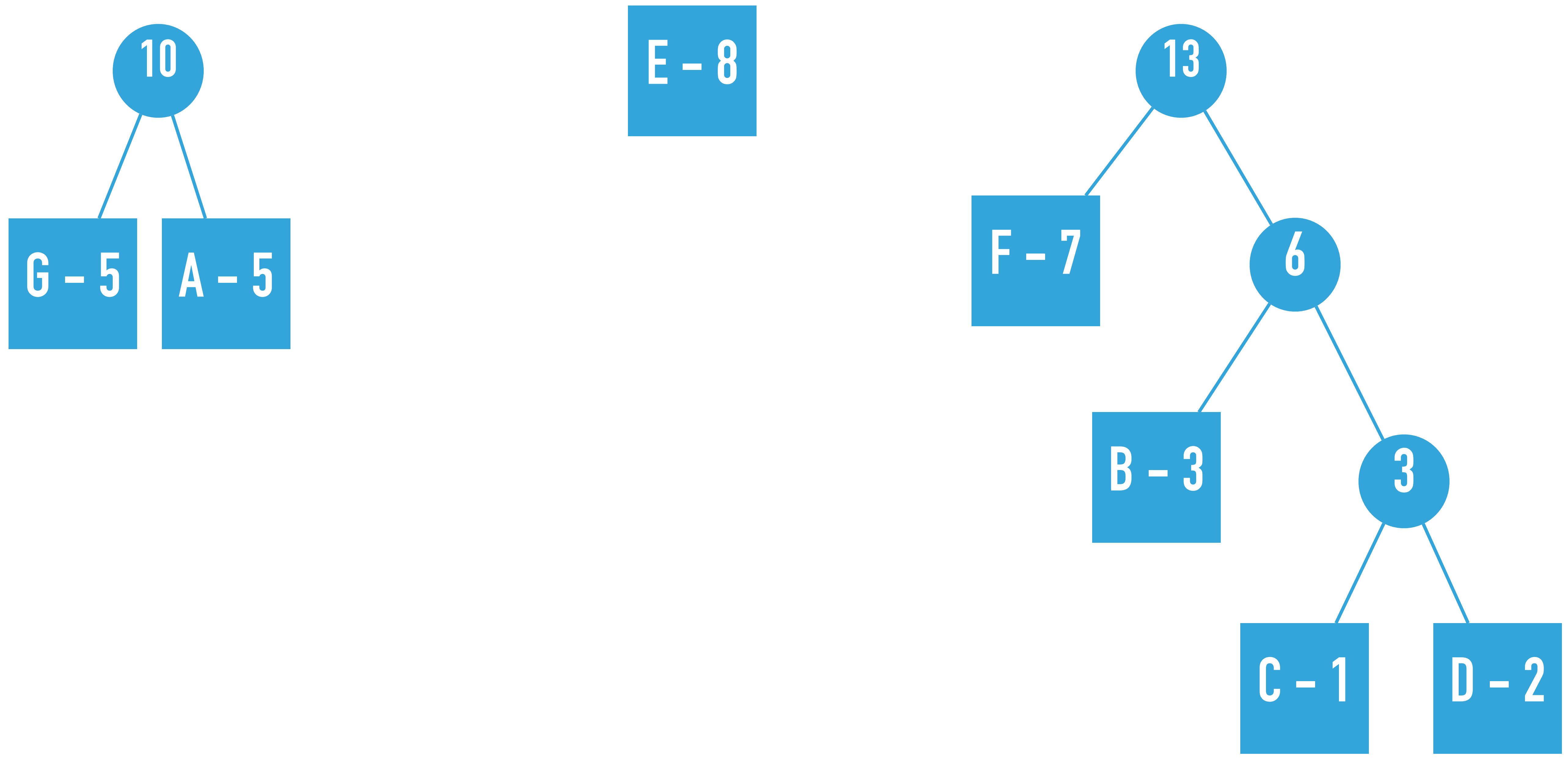
Question 5.1.10 Huffman



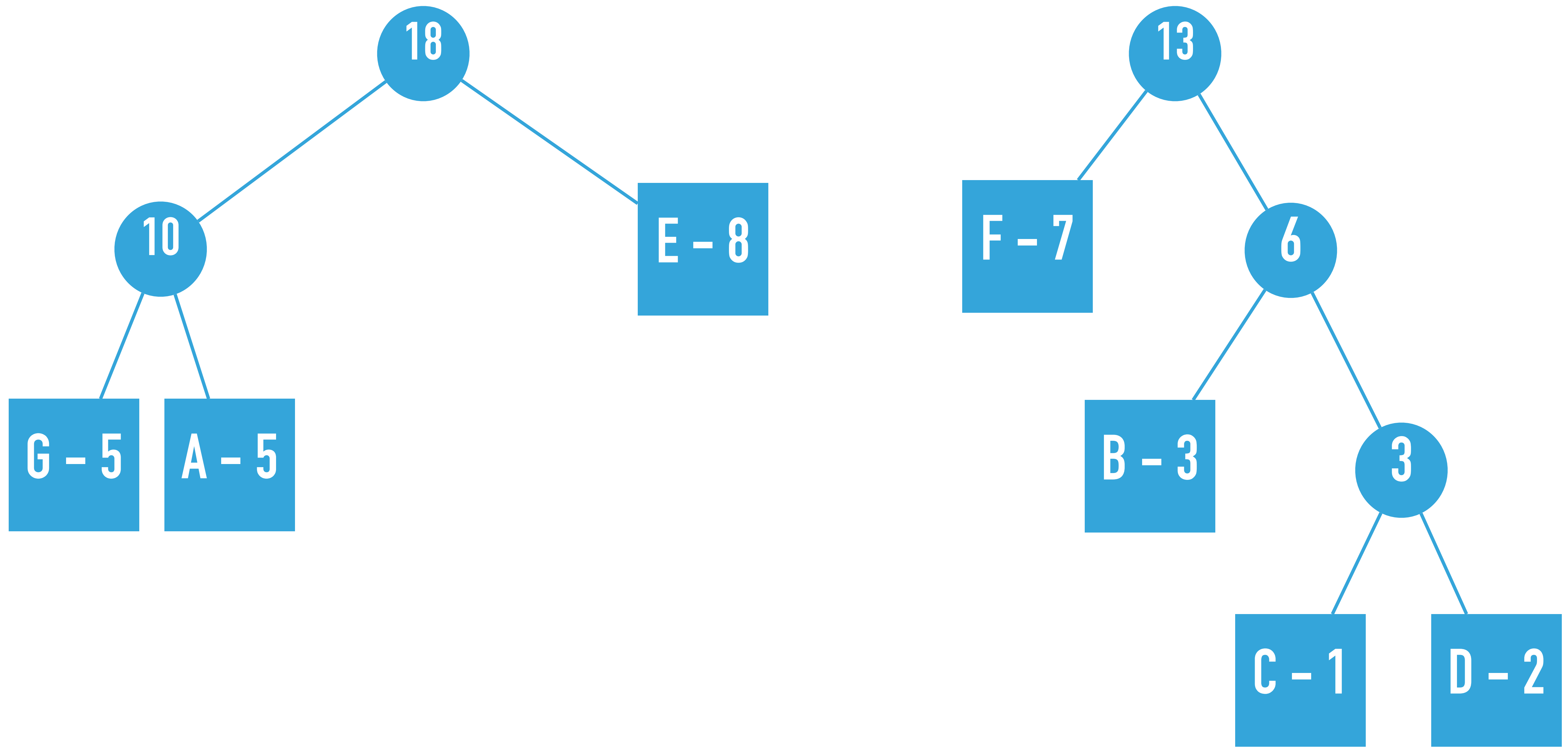
Question 5.1.10 Huffman



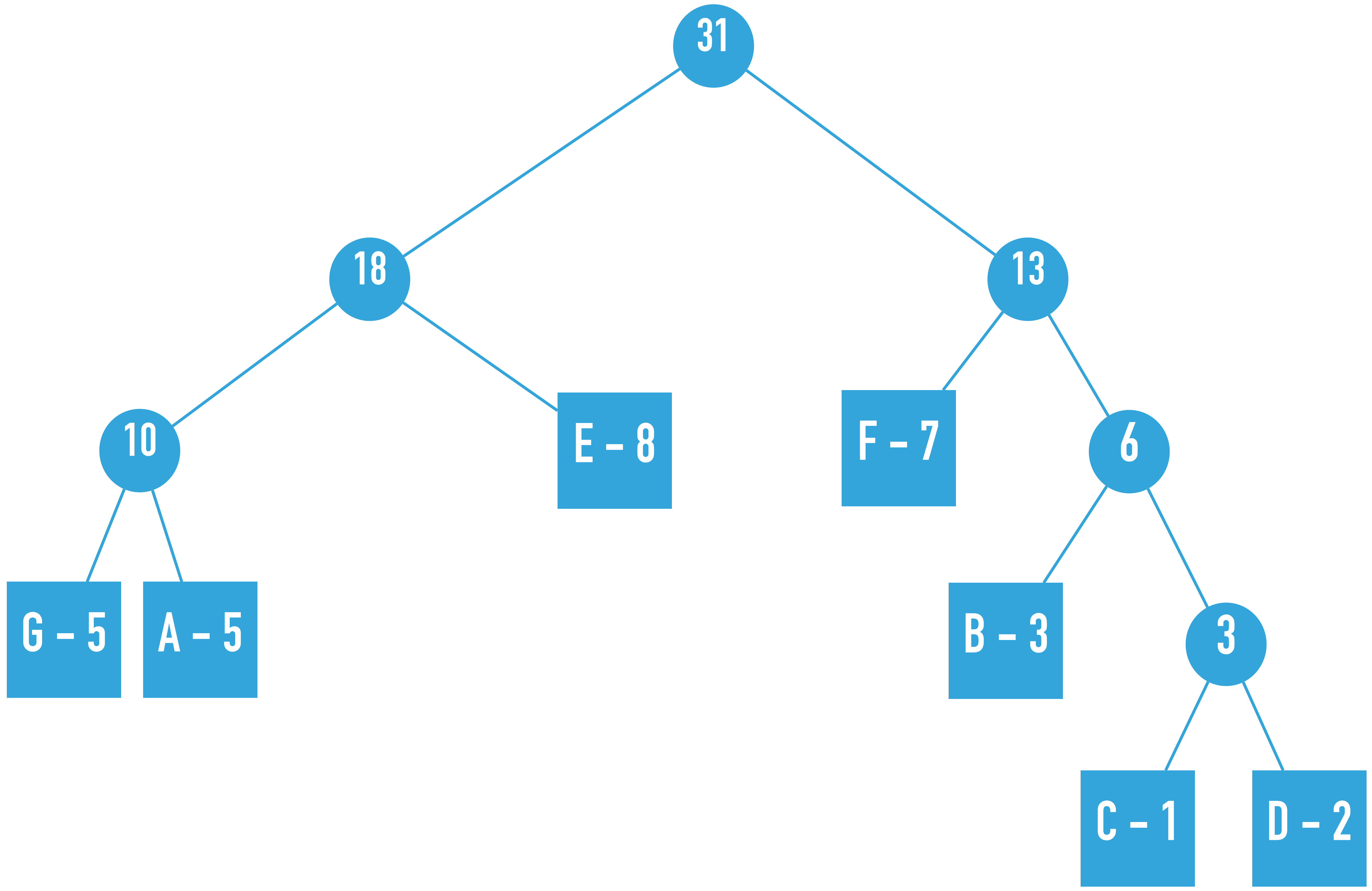
Question 5.1.10 Huffman



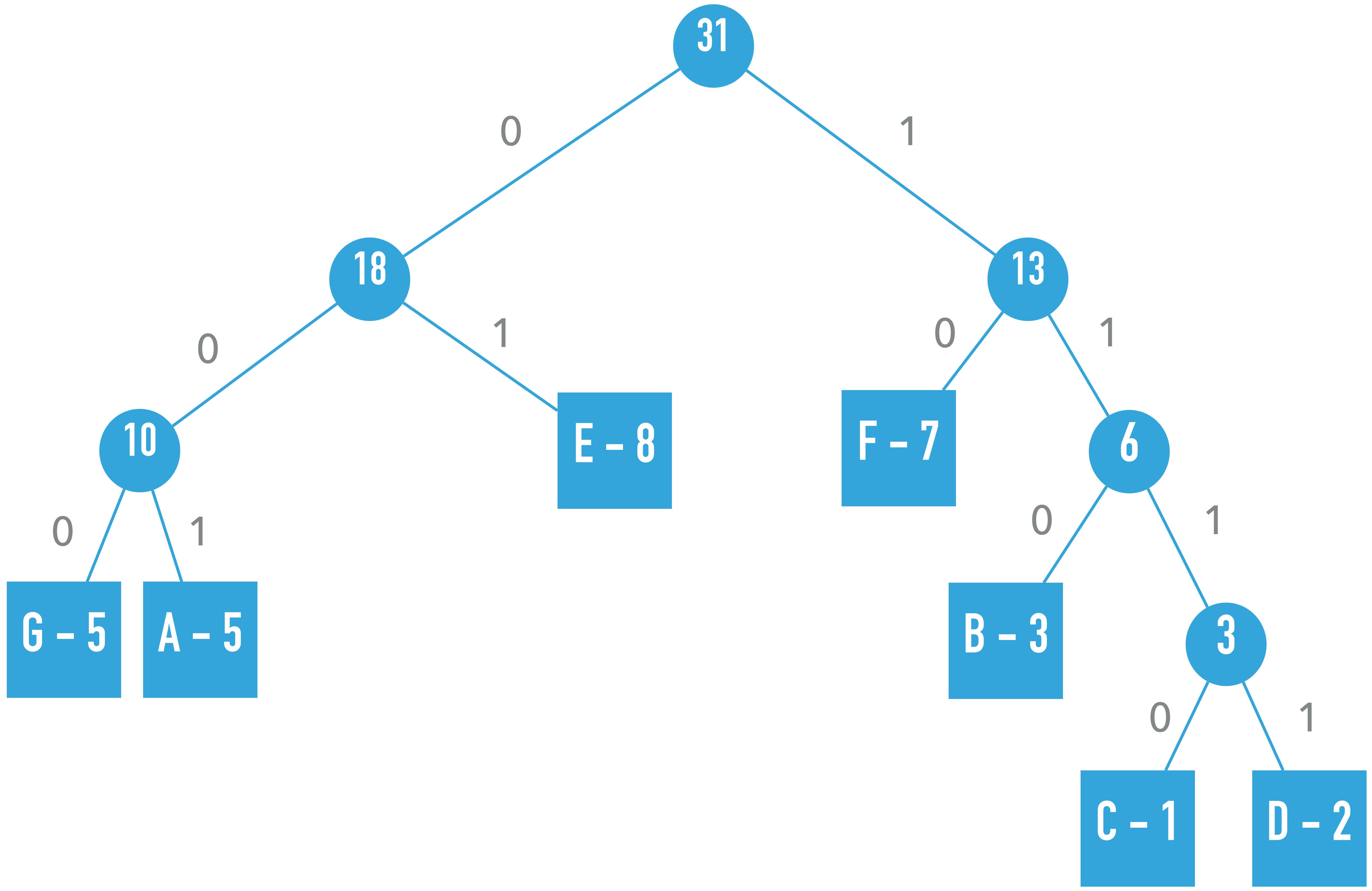
Question 5.1.10 Huffman



Question 5.1.10 Huffman



Question 5.1.10 Huffman



Question 5.1.10 Huffman

Calculons la complexité de Huffman avec des listes non triées:

- $\mathcal{O}(n)$ pour trouver les deux max
- $\Theta(1)$ pour l'union
- Ces deux étapes sont à répéter $n-1$ fois

$\mathcal{O}(n^2)$ au total

Pour des listes triées:

- $\Theta(1)$ pour trouver les deux max
- $\mathcal{O}(n)$ pour l'union (la replacer au bon endroit)
- Ces deux étapes sont à répéter $n-1$ fois

$\mathcal{O}(n^2)$ au total

Avec des heaps

- $\mathcal{O}(\log n)$ pour trouver les deux max
- $\mathcal{O}(\log n)$ pour l'union
- Ces deux étapes sont à répéter $n-1$ fois

$\mathcal{O}(n \log n)$ au total

Question 5.1.11 ETAPES du Huffman

1) Calculer l'arbre optimal:

- Parser le texte et calculer le nombre de lettre. Utilisation d'une (hash)map, $\mathcal{O}$(string.length)

- Effectuer l'algo vu précédemment. $\mathcal{O}(n \log n)$ où n est le nombre de caractères différents.

2) Compression:

- Enumérer tout les chemins de l'arbre pour créer un HashMap caractère vers suite de bits le représentant

- Pour chaque caractère, aller chercher dans le HashMap sont représentant.

3) Décompression:

- Pour chaque bit, descendre dans l'arbre en suivant la bonne direction.

- A chaque noeud terminal, output le caractère et retourner à la racine de l'arbre.