LINFO 1121

DATA STRUCTURES AND ALGORITHMS

UCLouvain

TP2: Tris et propriétés des ensembles triés

Question 2.1.1: Insert element in a sorted array

1	3	6	10	15	21	27	34	42	51

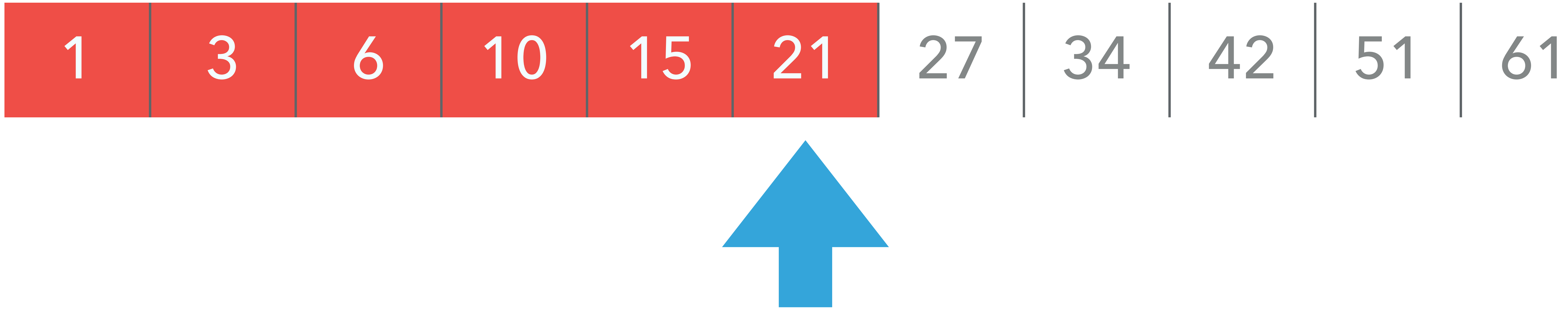
Where to insert 30?

Question 2.1.1: Insert element in a sorted array

1	3	6	10	15	21	27	34	42	51	61

21 < 30: on the right !

Question 2.1.1: Insert element in a sorted array



21 < 30: on the right !

Question 2.1.1: Insert element in a sorted array

1	3	6	10	15	21	27	34	42	51	61

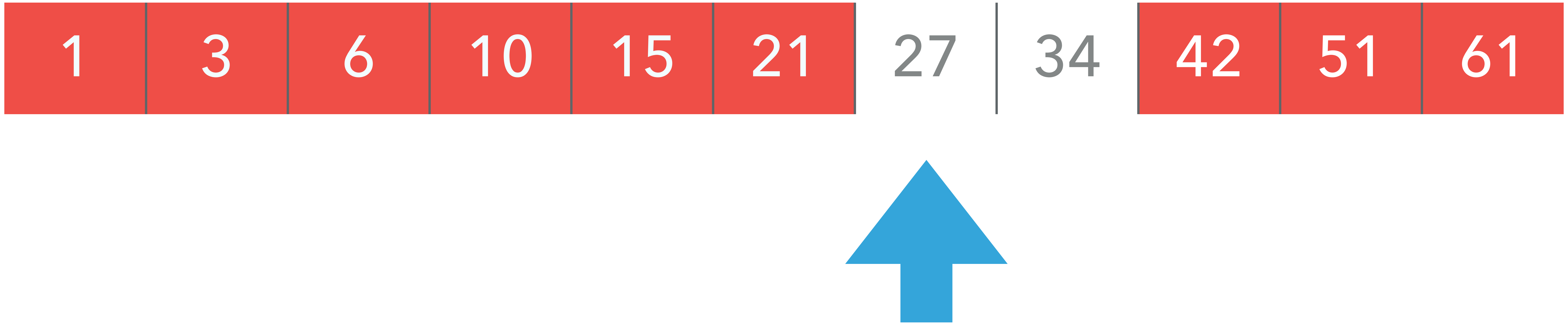
42 > 30: on the left !

Question 2.1.1: Insert element in a sorted array

1	3	6	10	15	21	27	34	42	51	61

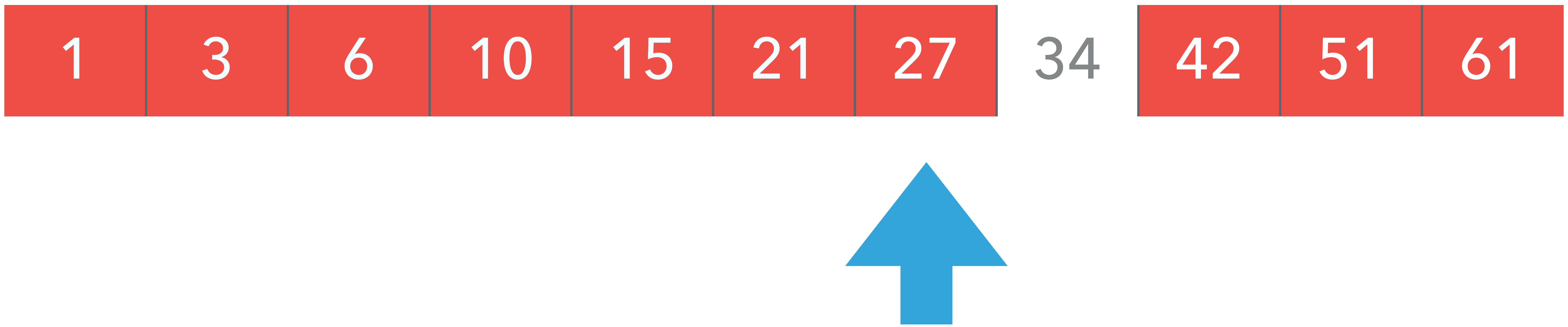
42 > 30: on the left !

Question 2.1.1: Insert element in a sorted array



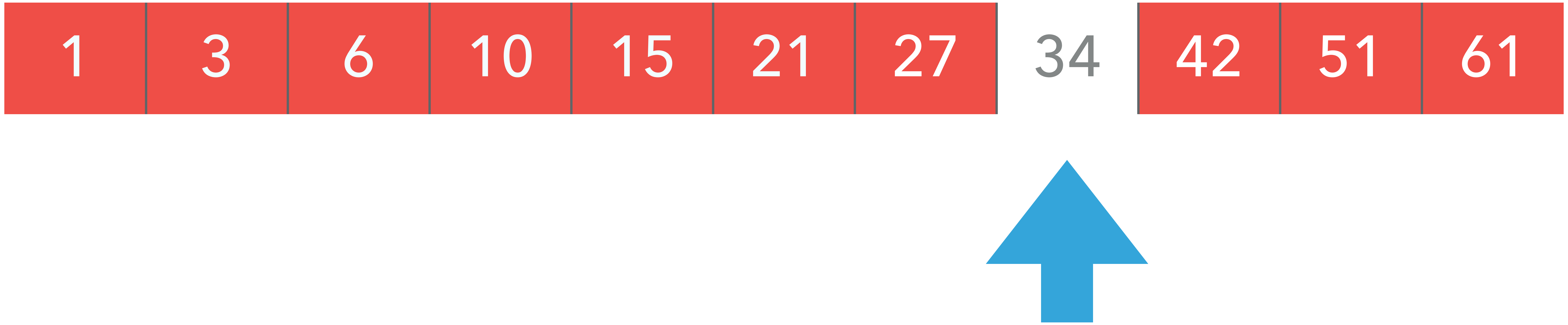
27 < 30: on the right !

Question 2.1.1: Insert element in a sorted array



27 < 30: on the right !

Question 2.1.1: Insert element in a sorted array



34 > 30: on the left !

Question 2.1.1: Insert element in a sorted array

What's the time complexity ?

$$f(n) = 1 + f(\frac{n}{2})$$
$$f(1) = 0$$

$$f(n) = \log_2 n$$

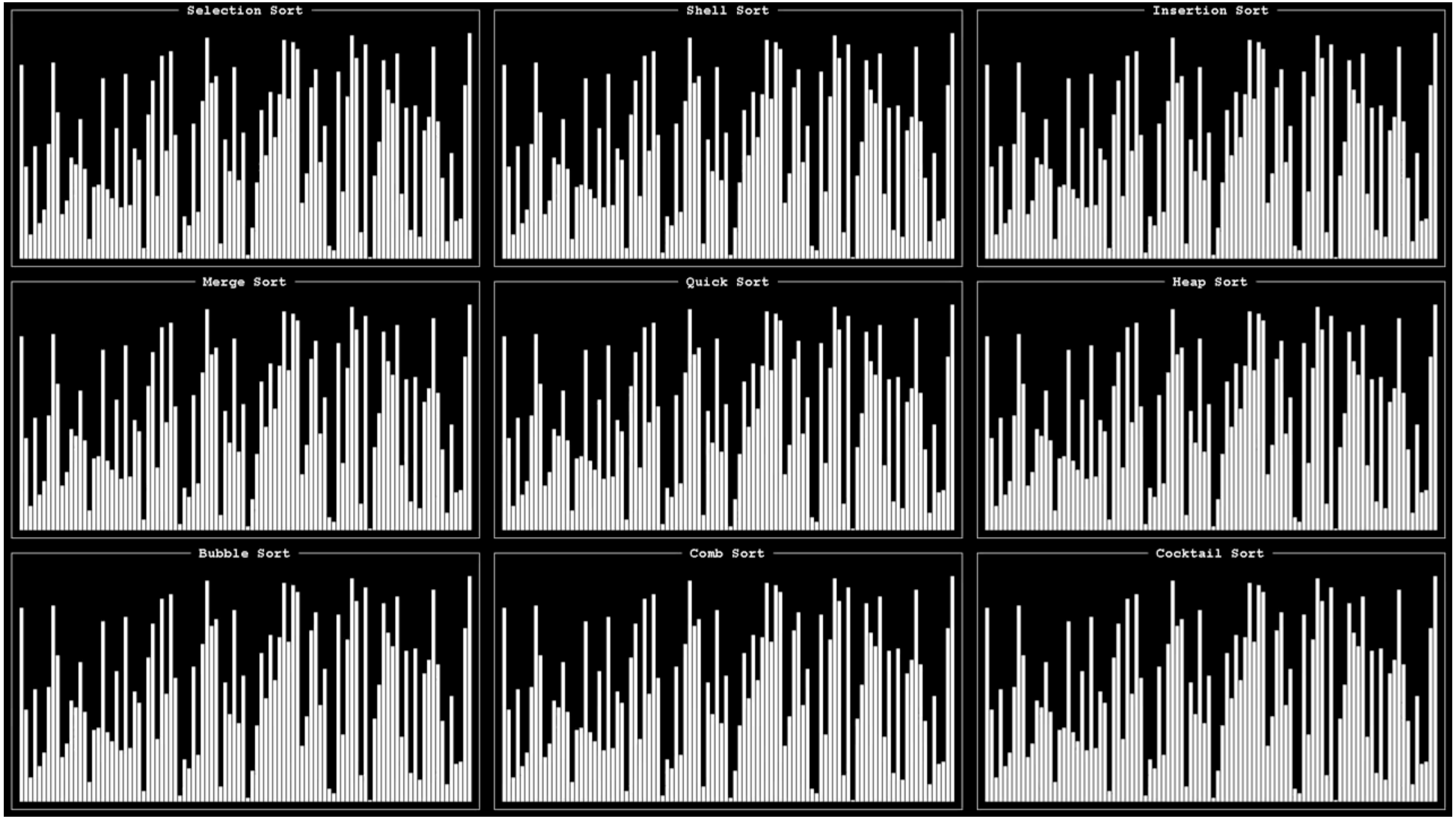
Question 2.1.1: Insert element in a sorted array

Can we do something faster than that ?

Yes, if we have information on the data distribution in the array !

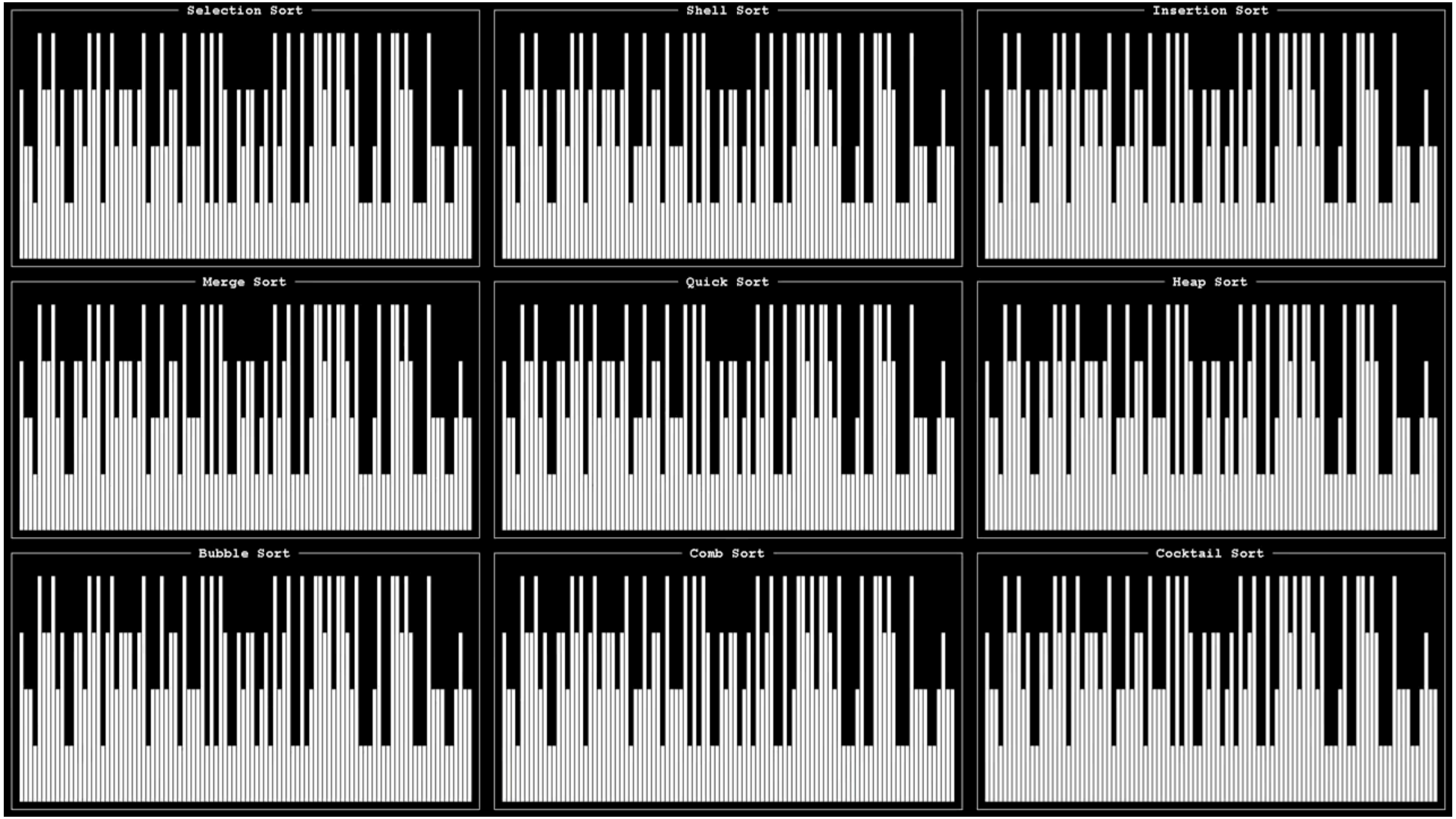
Sort algorithms, strengths and weaknesses

Strengths and weaknesses - Random



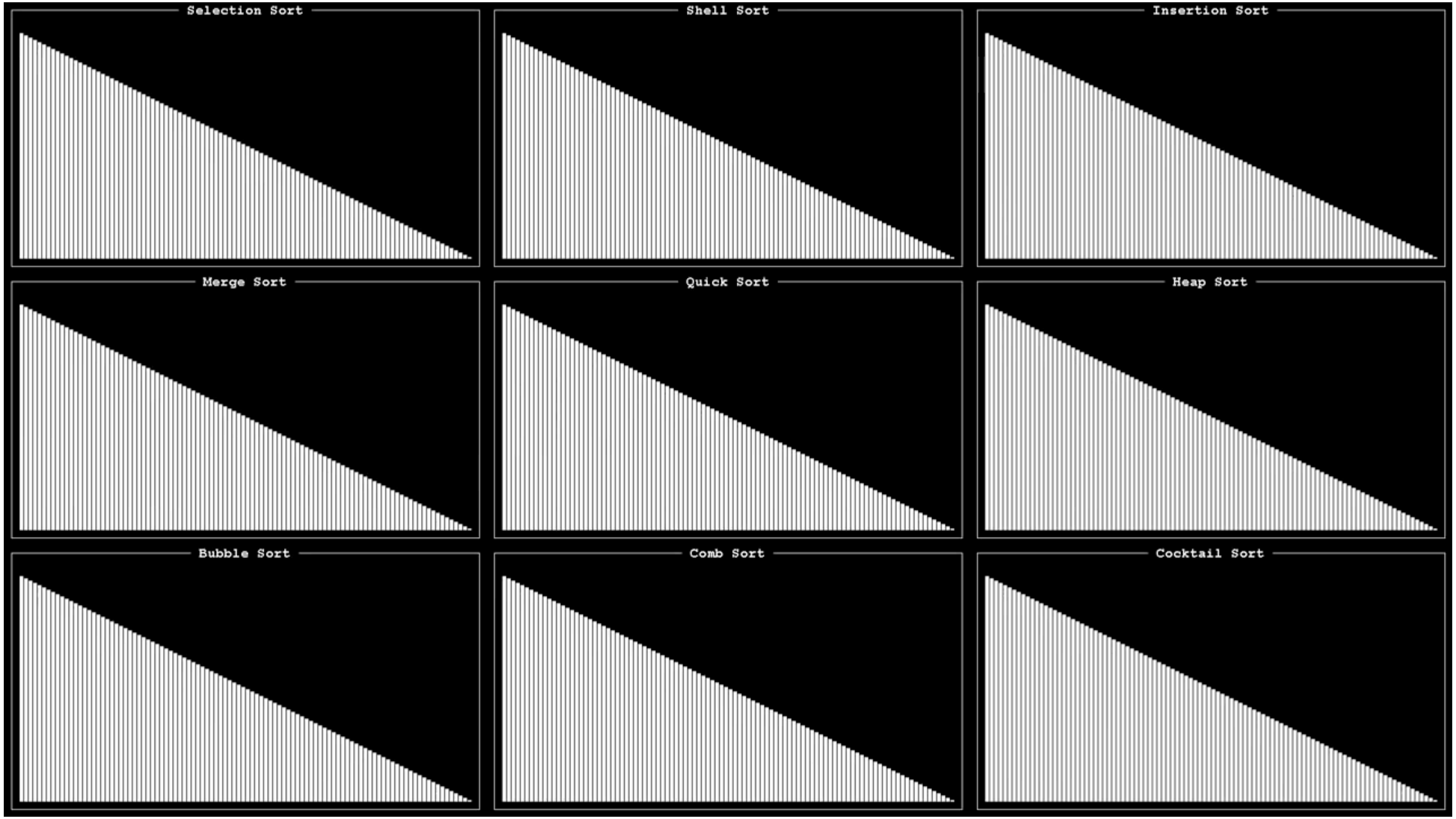
Sort algorithms, strengths and weaknesses

Strengths and weaknesses - Few unique



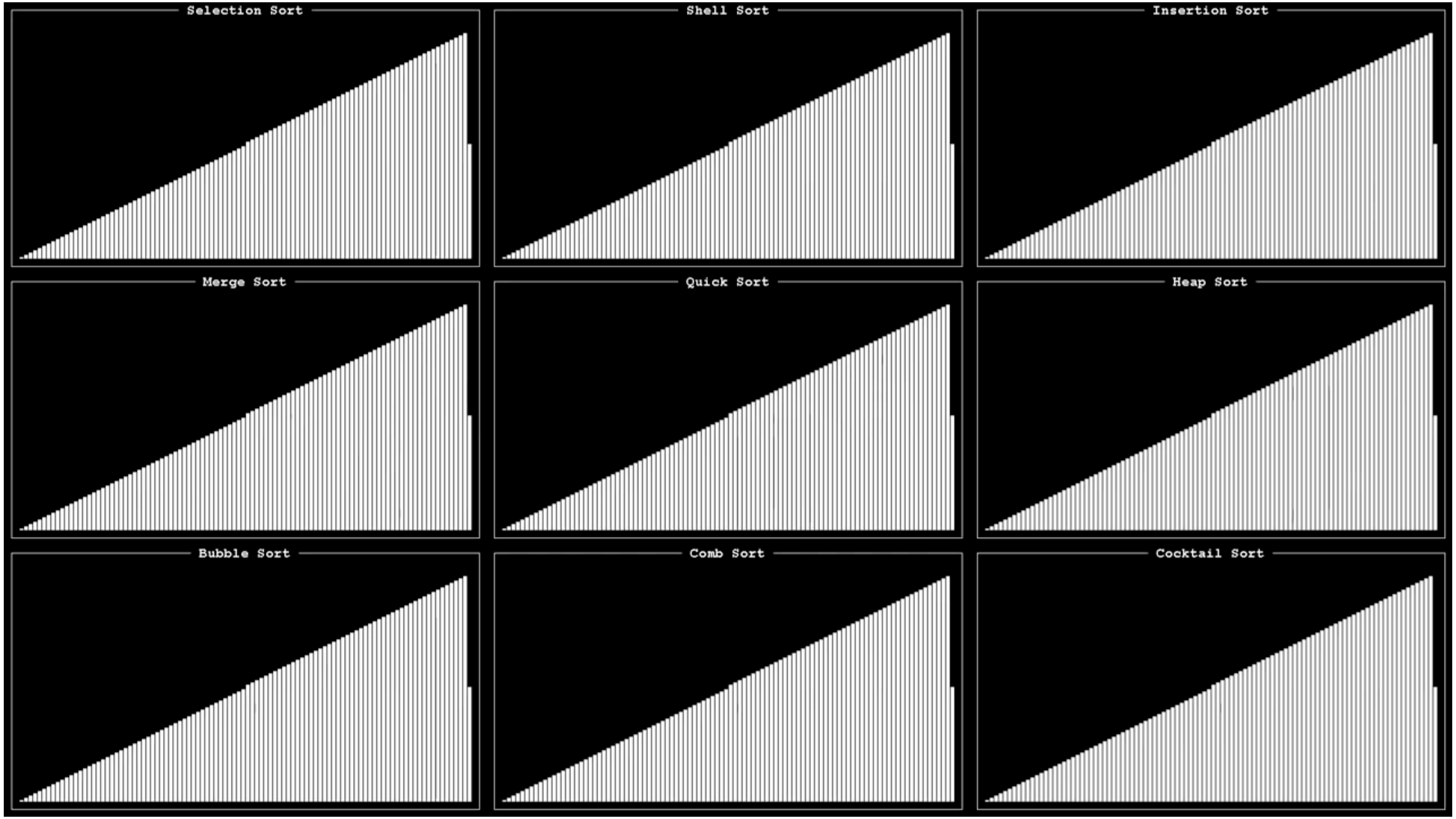
Sort algorithms, strengths and weaknesses

Strengths and weaknesses - Reversed



Sort algorithms, strengths and weaknesses

Strengths and weaknesses - Almost sorted



Question 2.1.2: Maximizing customer satisfaction

We consider the very general problem where we have n jobs to perform for clients and each job j takes tj seconds to complete. Only one job can be performed at a time.

The goal is to complete all jobs while maximizing customer satisfaction. Maximizing customer satisfaction means building a schedule that minimizes the average job completion time.

Example: if we have four jobs that take respectively 5, 8, 3, 1 seconds to finish then the order 1,2,3,4 takes an average completion time of (5 + 13 + 16 + 17) /4

Question 2.1.2: Maximizing customer satisfaction

$$\min \frac{\sum_{i=1}^{n} (\sum_{j=1}^{i} t_j)}{n}$$

Let's prove it by contradiction. Assume that we have a job A which is done just before B but takes more time than B

Supposed optimal

$$T_1 = (\sum_{i=1}^{A-1} t_i) + t_A + (\sum_{i=1}^{A-1} t_i) + t_A + t_B$$

Completion times if A and B are inverted

$$T_2 = (\sum_{i=1}^{B-1} t_i) + t_B + (\sum_{i=1}^{B-1} t_i) + t_B + t_A$$

Question 2.1.2: Maximizing customer satisfaction

$$T_1 = (\sum_{i=1}^{A-1} t_i) + t_A + (\sum_{i=1}^{A-1} t_i) + t_A + t_B \qquad T_2 = (\sum_{i=1}^{B-1} t_i) + t_B + (\sum_{i=1}^{B-1} t_i) + t_B + t_A$$

$$T_1 - T_2 = (\sum_{i=1}^{A-1} t_i) + t_A + (\sum_{i=1}^{A-1} t_i) + t_A + t_B - ((\sum_{i=1}^{B-1} t_i) + t_B + (\sum_{i=1}^{B-1} t_i) + t_B + t_A)$$

$$T_1 - T_2 = t_A + t_A + t_B - (t_B + t_B + t_A) = t_A - t_B > 0$$

By inversing A and B, we reduce the average time, we have a contradiction. Since this must hold

for every pair of job in the ordering, the jobs must be ordered by duration time.

Question 2.1.4 Cards sorter

How would you sort increasingly a pile of cards with the restriction that the only permitted operations are:

1. compare the first two cards,

1. exchange the first two cards,

3. move the first card to the back of the pile?

Question 2.1.4 Cards sorter

How would you sort increasingly a pile of cards with the restriction that the only permitted operations are:

1. compare the first two cards,

1. exchange the first two cards,

3. move the first card to the back of the pile?

Idea Try to maintain the **invariant** that the last i elements of the pile are sorted and those are the ith biggest ones.
After n iteration, the last n elements are sorted !

Question 2.1.4 Cards sorter

10	7	2	R	V	10	A	5	5	8

Top

Back

Question 2.1.4 Cards sorter

First iteration, let us find the largest element and put it at the end
!

10	7	2	R	V	10	A	5	5	8

Swap the card at the top to put the largest in second position

7	10	2	R	V	10	A	5	5	8

Put the front card at the end of the deck

10	2	R	V	10	A	5	5	8	7

Repeat until only one card is left, this is the largest. Put it at the end

A	7	2	10	V	R	5	5	8	7

7	2	10	V	R	5	5	8	7	A

Question 2.1.4 Cards sorter

What about the next iterations ?

7	2	10	V	R	5	5	8	7	A

Same process. Find the largest in the n-i first element, put it at the end !

R	A	2	7	10	V	5	5	8	7

2	7	10	V	5	5	8	7	R	A

Question 2.1.4 Cards sorter

```
for (int i = 0; i < n; i++) {
    // Invariant: the i last elements are sorted
    for (int k = 0; k < n; k++) {
        if (k <= n - 1 - i) {
            // put the smallest of the two top card on top
        }
        // move the top card at the end
    }
}
```

Question 2.1.5 Sorting a double linked list

How to sort a doubly linked list (which therefore does not allow access to a position by its index) efficiently? How complex is your algorithm?

Question 2.1.5 Sorting a double linked list

How to sort a doubly linked list (which therefore does not allow access to a position by its index) efficiently? How complex is your algorithm?

Can we take ideas from know sorting algorithms ?

Question 2.1.5 Sorting a double linked list

Merging two linked-list is similar to merging arrays in the MergeSort algorithm !

The "merge" operation can also be done in O(n+m) for linked list

```
public static void merge(Comparable[] a, int lo, int mid, int hi)
{  // Merge a[lo..mid] with a[mid+1..hi].
   int i = lo, j = mid+1;

   for (int k = lo; k <= hi; k++)  // Copy a[lo..hi] to aux[lo..hi].
      aux[k] = a[k];

   for (int k = lo; k <= hi; k++)  // Merge back to a[lo..hi].
      if      (i > mid)              a[k] = aux[j++];
      else if (j > hi )              a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                           a[k] = aux[i++];
}
```

Question 2.1.6 Number of unordered pairs

Design an efficient algorithm for counting the number of pairs of disordered values. For example in the sequence 1,3,2,5,6,4,8 there are the pairs (3,2),(5,4),(6,4) which are unordered. Justify the complexity of your algorithm and give its pseudo code.

Question 2.1.6 Number of unordered pairs

Design an efficient algorithm for counting the number of pairs of disordered values. For example in the sequence 1,3,2,5,6,4,8 there are the pairs (3,2),(5,4),(6,4) which are unordered. Justify the complexity of your algorithm and give its pseudo code.

Hint: Assume two arrays A and B, let A.B be the array result of the concatenation of A and B. Let nUnsorted(A) be the number of unsorted pairs in an array A.
We have the following property that you can prove:

$$\text{nUnsorted}(A.B)=\text{nUnsorted}(A)+\text{nUnsorted}(B)+|\{(i,j):A[i]>B[j]\}|$$

Question 2.1.6 Number of unordered pairs

Computing the unsorted elements in A and B is linear if the two arrays are sorted

```
int wrongOrder(int[] A, int [] B) {
    // A et B sont des tableaux triés dans l'ordre croissant
    int posB = B.length;
    int count = 0;
    for(int i = A.length - 1; i >= 0; i--) {
        while(posB != 0 && B[posB-1] >= A[i])
            posB--;
        count += posB;
    }
    return count;
}
```

Question 2.1.6 Number of unordered pairs

```
public static int numberUnsortedPairs(int [] array, int lo, int hi) {
    if (lo <= hi) return;
    int mid = (lo + hi)/2;
    int nA = numberUnsortedPairs(array, lo, mid);
    int nB = numberUnsortedPairs(array, mid+1 , hi);
    int wab = wrongOrder(array, lo, mid, hi);
    merge(array, lo, mid, hi);
}
```

Question 2.1.7 COMPARABLE/COMPARATOR

Imagine that we want to sort a collection of Person objects lexicographically by their (weight, age, height) but also Student objects by their (age, grade, year), how to avoid duplicating the sorting algorithm specifically for these classes?

Explain why the notions of Comparable and Comparator of Java are useful for this? Explain how you would implement an efficient Comparator for String.

Question 2.1.8 Stable sort from an unstable one ?

Is it possible to get a stable sort starting from an unstable sorting algorithm? How?

Question 2.1.9 Find the third largest value

How to find the third largest value in an array ?

Question 2.1.9 Find the third largest value

How to find the third largest value in an array ? We can use the same algorithm as to find the minimum, it is even linear !

Question 2.1.9 Find the third largest value

How to find the third largest value in an array ? We can use the same algorithm as to find the minimum, it is even linear !

```
public static int findThirdLargest(int [] array) {
    int max1, max2, max3 = Integer.MIN_VALUE;
    for (Integer i : array) {
        if (i > max1) {
            max3 = max2; max2 = max1; max1 = i;
        } else if (i > max2) {
            max3 = max2; max2 = i;
        } else if (i > max3) {
            max3 = i;
        }
    }
    return max3;
}
```

Question 2.1.9 Find the third largest value

What happen if now we want a generic method to find the n-th largest value ?

Question 2.1.9 Find the third largest value

What happen if now we want a generic method to find the n-th largest value ?

```
public static int findNLargest(int [] array, int n) {
    Arrays.sort(array);
    return array[n];
}
```

(Assuming there are no duplicate, but in case of duplicate a linear pass over the array is doable and still O(n log(n)))

Question 2.1.10 Median

How would you get the median of an array of values (so the n/2 th value)? What is the time complexity of your algorithm?

Question 2.1.10 Median

How would you get the median of an array of values (so the n/2 th value)? What is the time complexity of your algorithm?

We can sort the array, and then take the element at the middle index. Complexity is O(n log(n)), good enough. Can we do better ?

Question 2.1.10 Median

What does the partition function do?

```
public class Quick
{
   public static void sort(Comparable[] a)
   {
      StdRandom.shuffle(a);          // Eliminate dependence on input.
      sort(a, 0, a.length - 1);
   }

   private static void sort(Comparable[] a, int lo, int hi)
   {
      if (hi <= lo) return;
      int j = partition(a, lo, hi);  // Partition (see page 291).
      sort(a, lo, j-1);              // Sort left part a[lo .. j-1].
      sort(a, j+1, hi);              // Sort right part a[j+1 .. hi].
   }
}
```

Question 2.1.10 Median

```
public static int median(int[] array) {
    int lo = 0;
    int hi = array.length;
    int i = lo;
    while (i != array.length / 2) {
        int i = partition(array, lo, hi);
        if i < array.length / 2 {
            lo = i;
        } else if i > array.length / 2 {
            hi = i;
        }
    }
}
```

Question 2.1.11 AUTOBOXING, UNBOXING

What is Autoboxing and Unboxing in Java? How can this impact the performance of a sorting algorithm?