## Week 03: Selection

FanFly

March 22, 2020

First, let us introduce the selction problem.

FanFly Week 03: Selection

First, let us introduce the selction problem.

#### The Selection Problem

- Input: An array A of n numbers and an index k with  $0 \le k < n$ .
- Output: The element at index k in the sorted array reordered from A.

◆ロト ◆個ト ◆ 恵ト ◆ 恵 ・ かくで

FanFly Week 03: Selection

First, let us introduce the selction problem.

#### The Selection Problem

- Input: An array A of n numbers and an index k with  $0 \le k < n$ .
- Output: The element at index k in the sorted array reordered from A.

Note that the output is the (k + 1)-th smallest number of A, which is the (k + 1)-th **order statistic** of A.

FanFly

First, let us introduce the selction problem.

#### The Selection Problem

- Input: An array A of n numbers and an index k with  $0 \le k < n$ .
- Output: The element at index k in the sorted array reordered from A.

Note that the output is the (k + 1)-th smallest number of A, which is the (k + 1)-th **order statistic** of A.

• It is useful in many cases, e.g., the  $\lfloor (n-1)/2 \rfloor$ -th order statistic is the **lower median**.

◆ロト ◆母 ト ◆ 差 ト ◆ 差 ・ 釣 Q ○

FanFly Week 03: Selection March 22, 2020 2 / 16

One can sort the array and then output the element at index k.

```
def sort_and_select(A, k):
    B = sorted(A)
    return B[k]
```

FanFly Week 03: Selection March 22, 2020 3/16

One can sort the array and then output the element at index k.

```
def sort_and_select(A, k):
    B = sorted(A)
    return B[k]
```

What is its time complexity?



One can sort the array and then output the element at index k.

```
def sort_and_select(A, k):
    B = sorted(A)
    return B[k]
```

What is its time complexity?

• One needs  $\Theta(n \log n)$  time to sort an array with length n.

One can sort the array and then output the element at index k.

```
def sort_and_select(A, k):
    B = sorted(A)
    return B[k]
```

What is its time complexity?

- One needs  $\Theta(n \log n)$  time to sort an array with length n.
- Indexing takes time  $\Theta(1)$ .

One can sort the array and then output the element at index k.

```
def sort_and_select(A, k):
    B = sorted(A)
    return B[k]
```

What is its time complexity?

- One needs  $\Theta(n \log n)$  time to sort an array with length n.
- Indexing takes time  $\Theta(1)$ .

Thus, this algorithm runs in  $\Theta(n \log n)$  time.

If k is small, we can find only the (k+1) smallest numbers.

```
def partial_sort_and_select(A, k):
    B = list(A)
    for i in range(k + 1):
        for j in range(i + 1, len(A)):
             if B[j] < B[i]:</pre>
                 B[i], B[j] = B[j], B[i]
    return B[k]
```

If k is small, we can find only the (k+1) smallest numbers.

```
def partial_sort_and_select(A, k):
    B = list(A)
    for i in range(k + 1):
        for j in range(i + 1, len(A)):
            if B[j] < B[i]:
                B[i], B[j] = B[j], B[i]
    return B[k]</pre>
```

What is its time complexity?

If k is small, we can find only the (k+1) smallest numbers.

```
def partial_sort_and_select(A, k):
    B = list(A)
    for i in range(k + 1):
        for j in range(i + 1, len(A)):
            if B[j] < B[i]:
            B[i], B[j] = B[j], B[i]
    return B[k]</pre>
```

What is its time complexity?

• Swaping two elements takes  $\Theta(1)$  time.

If k is small, we can find only the (k+1) smallest numbers.

```
def partial_sort_and_select(A, k):
    B = list(A)
    for i in range(k + 1):
        for j in range(i + 1, len(A)):
            if B[j] < B[i]:
            B[i], B[j] = B[j], B[i]
    return B[k]</pre>
```

What is its time complexity?

- Swaping two elements takes  $\Theta(1)$  time.
- The number of the inner iterations is

$$(n-1)+(n-2)+\cdots+(n-k-1)=\left(n-\frac{k+2}{2}\right)(k+1).$$

- 4 ロ ト 4 個 ト 4 種 ト 4 種 ト - 種 - か Q (^)

If k is small, we can find only the (k+1) smallest numbers.

```
def partial_sort_and_select(A, k):
    B = list(A)
    for i in range(k + 1):
        for j in range(i + 1, len(A)):
             if B[j] < B[i]:</pre>
                 B[i], B[j] = B[j], B[i]
    return B[k]
```

What is its time complexity?

- Swaping two elements takes  $\Theta(1)$  time.
- The number of the inner iterations is

$$(n-1)+(n-2)+\cdots+(n-k-1)=\left(n-\frac{k+2}{2}\right)(k+1).$$

Thus, this algorithm runs in  $\Theta(nk)$  time.

Let us compare the two algorithms.



FanFly Week 03: Selection

Let us compare the two algorithms.

• The first attempt runs in  $\Theta(n \log n)$  time.

Week 03: Selection

Let us compare the two algorithms.

- The first attempt runs in  $\Theta(n \log n)$  time.
- The second attempt runs in  $\Theta(nk)$  time.



FanFly

Let us compare the two algorithms.

- The first attempt runs in  $\Theta(n \log n)$  time.
- The second attempt runs in  $\Theta(nk)$  time.

Which one is faster?



5/16

FanFly Week 03: Selection March 22, 2020

Let us compare the two algorithms.

- The first attempt runs in  $\Theta(n \log n)$  time.
- The second attempt runs in  $\Theta(nk)$  time.

Which one is faster? It depends on the size of k.



FanFly

Let us compare the two algorithms.

- The first attempt runs in  $\Theta(n \log n)$  time.
- The second attempt runs in  $\Theta(nk)$  time.

Which one is faster? It depends on the size of k.

• If k is large (i.e.,  $k = \Omega(\log n)$ ), we can choose the first method.



FanFly

Let us compare the two algorithms.

- The first attempt runs in  $\Theta(n \log n)$  time.
- The second attempt runs in  $\Theta(nk)$  time.

Which one is faster? It depends on the size of k.

- If k is large (i.e.,  $k = \Omega(\log n)$ ), we can choose the first method.
- If k is small (i.e.,  $k = O(\log n)$ ), we can choose the second method.

4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶
4□▶

Let us compare the two algorithms.

- The first attempt runs in  $\Theta(n \log n)$  time.
- The second attempt runs in  $\Theta(nk)$  time.

Which one is faster? It depends on the size of k.

- If k is large (i.e.,  $k = \Omega(\log n)$ ), we can choose the first method.
- If k is small (i.e.,  $k = O(\log n)$ ), we can choose the second method.

Thus, we have an algorithm that runs in  $\Theta(\min\{n \log n, nk\})$  time.

◆ロト ◆個ト ◆ 恵ト ◆ 恵 ・ かくで

However, there exists faster algorithms that solve the selction problem.



FanFly

However, there exists faster algorithms that solve the selction problem.

• A deterministic algorithm, called the **median of medians** algorithm, solves the problem in O(n) time.



FanFly Week 03: Selection March 22, 2020 6/16

However, there exists faster algorithms that solve the selction problem.

- A deterministic algorithm, called the median of medians algorithm, solves the problem in O(n) time.
- A randomized algorithm, called the quick select algorithm, solves the problem in expected O(n) time.



FanFly Week 03: Selection March 22, 2020 6/16

However, there exists faster algorithms that solve the selction problem.

- A deterministic algorithm, called the median of medians algorithm, solves the problem in O(n) time.
- A randomized algorithm, called the quick select algorithm, solves the problem in expected O(n) time.

We are going to introduce the latter one, and thus we need to deal with randomization first.

FanFly Week 03: Selection March 22, 2020 6/16

We use the package random to generate pseudo-random numbers.

◄□▶◀圖▶◀불▶◀불▶ 불 쒸٩○

We use the package random to generate pseudo-random numbers.

>>> import random



FanFly

We use the package random to generate pseudo-random numbers.

```
>>> import random
>>> random.random()  # generate number in [0.0, 1.0)
0.5476045155149599
```



7/16

FanFly Week 03: Selection March 22, 2020

We use the package random to generate pseudo-random numbers.

```
>>> import random

>>> random.random()  # generate number in [0.0, 1.0)
0.5476045155149599
>>> random.random()
0.630346861646684
```

We use the package random to generate pseudo-random numbers.

```
>>> import random
>>> random.random()
                       # generate number in [0.0, 1.0)
0.5476045155149599
>>> random.random()
0.630346861646684
>>> random.randrange(5)
                          # generate integer in [0, 5)
```

7/16

FanFly Week 03: Selection March 22, 2020

We use the package random to generate pseudo-random numbers.

```
>>> import random
>>> random.random()
                       # generate number in [0.0, 1.0)
0.5476045155149599
>>> random.random()
0.630346861646684
>>> random.randrange(5)
                          # generate integer in [0, 5)
>>> random.randrange(5)
4
```

FanFly Week 03: Selection March 22, 2020 7/16

We use the package random to generate pseudo-random numbers.

```
>>> import random
>>> random.random()
                       # generate number in [0.0, 1.0)
0.5476045155149599
>>> random.random()
0.630346861646684
>>> random.randrange(5)
                           # generate integer in [0, 5)
>>> random.randrange(5)
>>> random.randrange(5)
3
```

We use the package random to generate pseudo-random numbers.

```
>>> import random
>>> random.random()
                       # generate number in [0.0, 1.0)
0.5476045155149599
>>> random.random()
0.630346861646684
>>> random.randrange(5)
                           # generate integer in [0, 5)
>>> random.randrange(5)
>>> random.randrange(5)
3
```

See the official documentation page for more examples.

### **Partition**

Now we introduce the partition algorithm, which is really useful.

FanFly

### **Partition**

Now we introduce the partition algorithm, which is really useful.

• First, we choose the last element as **pivot**.



Week 03: Selection FanFly March 22, 2020 8/16

### **Partition**

Now we introduce the partition algorithm, which is really useful.

- First, we choose the last element as **pivot**.
- Then we reorder the array such that elements less than pivot come before the pivot, while elements greater than pivot come after the pivot.

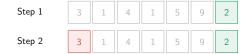


### **Partition**

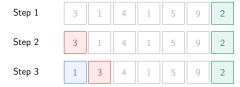
Now we introduce the partition algorithm, which is really useful.

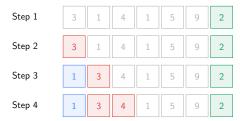
- First, we choose the last element as **pivot**.
- Then we reorder the array such that elements less than pivot come before the pivot, while elements greater than pivot come after the pivot.

Step 1

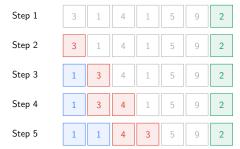


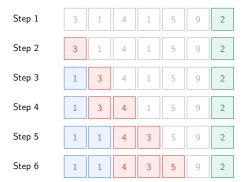
9/16





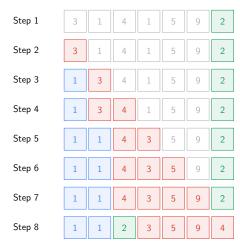
9/16







9/16



9/16

### Quick Select

With the partition procedure, one can select the any order statistic really quickly!

```
def quick_select(A, 1, r, k):
    i = partition(A, 1, r)
    if k == i:
        return A[k]
    elif k > i:
        return quick_select(A, i + 1, r, k)
    else:
        return quick_select(A, 1, i, k)
```

FanFly

Let us analyze the time complexity of quick select.



FanFly Week 03: Selection March 22, 2020 11 / 16

Let us analyze the time complexity of quick select.

• The partition procedure runs in  $\Theta(n)$  time when the array has n elements.



FanFly Week 03: Selection March 22, 2020 11 / 16

Let us analyze the time complexity of quick select.

- The partition procedure runs in  $\Theta(n)$  time when the array has n elements.
- If the partition is balanced, then we have

$$T(n) = T(n/2) + \Theta(n)$$

for n > 2, implying  $T(n) = \Theta(n)$ .



FanFly Week 03: Selection March 22, 2020 11 / 16

Let us analyze the time complexity of quick select.

- The partition procedure runs in  $\Theta(n)$  time when the array has n elements.
- If the partition is balanced, then we have

$$T(n) = T(n/2) + \Theta(n)$$

for n > 2, implying  $T(n) = \Theta(n)$ .

However, if the partition is unbalanced, then we may have

$$T(n) = T(n-1) + \Theta(n)$$

for  $n \ge 2$ , implying  $T(n) = \Theta(n^2)$ .

FanFly

## Analysis of Quick Select (cont.)

If we'd like to make the partition balanced, we can use some tricks.



FanFly Week 03: Selection March 22, 2020 12 / 16

## Analysis of Quick Select (cont.)

If we'd like to make the partition balanced, we can use some tricks.

 Instead of choosing the last element as pivot, we can choose a random element as pivot.



FanFly Week 03: Selection March 22, 2020 12 / 16

## Analysis of Quick Select (cont.)

If we'd like to make the partition balanced, we can use some tricks.

- Instead of choosing the last element as pivot, we can choose a random element as pivot.
- With randomization, the partition is somehow balanced in most cases, and it leads to a expected linear time complexity.



FanFly Week 03: Selection March 22, 2020 12 / 16

### Conclusion

The time complexity of the selction problem is  $\Theta(n)$ .

#### The Selection Problem

- Input: An array A of n numbers and an index k with  $0 \le k < n$ .
- Output: The element at index k in the sorted array reordered from A.

March 22, 2020

13 / 16

# Conclusion

The time complexity of the selction problem is  $\Theta(n)$ .

#### The Selection Problem

- Input: An array A of n numbers and an index k with  $0 \le k < n$ .
- ullet Output: The element at index k in the sorted array reordered from A.

The comparison among the algorithms that solves the selction problem is as follows.

Algorithm	Worst-case Time Complexity
Sorting (Merge Sort)	$\Theta(n \log n)$
Partial Selection Sort	$\Theta(nk)$
Quick Select	$\Theta(n)$ (expected)
Median of Medians	$\Theta(n)$

FanFly Week 03: Selection March 22, 2020 13/16

If T(n) satisfies

$$T(n) = \begin{cases} O(1), & \text{if } n \leq 1 \\ T(n-2) + \Theta(n), & \text{otherwise}, \end{cases}$$

what is the exact complexity of T(n)?



FanFly Week 03: Selection March 22, 2020 14 / 16

If T(n) satisfies

$$T(n) = \begin{cases} O(1), & \text{if } n \leq 1 \\ T(n-2) + \Theta(n), & \text{otherwise}, \end{cases}$$

what is the exact complexity of T(n)?

#### Solution

We have  $T(n) = \Theta(n^2)$ .



14 / 16

FanFly Week 03: Selection March 22, 2020

If T(n) satisfies

$$T(n) = \begin{cases} O(1), & \text{if } n \leq 1 \\ T(9n/10) + \Theta(n), & \text{otherwise,} \end{cases}$$

what is the exact complexity of T(n)?



FanFly Week 03: Selection March 22, 2020 15 / 16

If T(n) satisfies

$$T(n) = \begin{cases} O(1), & \text{if } n \leq 1 \\ T(9n/10) + \Theta(n), & \text{otherwise,} \end{cases}$$

what is the exact complexity of T(n)?

#### Solution

Use the master theorem, and we have  $T(n) = \Theta(n)$ .



FanFly

In fact, there is an algorithm called quick sort, that also uses the partition procedure.



FanFly Week 03: Selection March 22, 2020 16 / 16

In fact, there is an algorithm called quick sort, that also uses the partition procedure.

```
def quick_sort(A, 1, r):
    if r - 1 <= 1:
        return
    i = partition(A, 1, r)
    quick_sort(A, l, i)
    quick_sort(A, i + 1, r)
```



16 / 16

FanFly Week 03: Selection March 22, 2020

In fact, there is an algorithm called quick sort, that also uses the partition procedure.

```
def quick_sort(A, 1, r):
    if r - 1 <= 1:
        return
    i = partition(A, 1, r)
    quick_sort(A, 1, i)
    quick_sort(A, i + 1, r)</pre>
```

If you are really unlucky such that the partition is unbalanced in each iteration, how much time will it takes to run quick sort?

In fact, there is an algorithm called quick sort, that also uses the partition procedure.

```
def quick_sort(A, 1, r):
    if r - 1 <= 1:
        return
    i = partition(A, 1, r)
    quick_sort(A, 1, i)
    quick_sort(A, i + 1, r)
```

If you are really unlucky such that the partition is unbalanced in each iteration, how much time will it takes to run quick sort?

You can assume that the pivot is always the largest element.

In fact, there is an algorithm called quick sort, that also uses the partition procedure.

```
def quick_sort(A, 1, r):
    if r - 1 <= 1:
        return
    i = partition(A, 1, r)
    quick_sort(A, 1, i)
    quick_sort(A, i + 1, r)
```

If you are really unlucky such that the partition is unbalanced in each iteration, how much time will it takes to run quick sort?

You can assume that the pivot is always the largest element.

#### Solution

It takes  $\Theta(n^2)$  time.

(However, quick sort can run in expected  $\Theta(n \log n)$  time if randomization is used.)

> FanFly Week 03: Selection March 22, 2020 16 / 16