# Lecture 4: The Linear Time Selection

## Selection Problem

Given a sequence of numbers $\langle a_1, \ldots, a_n \rangle$, and an integer $i$, $1 \leq i \leq n$, find the $i$th smallest element. When $i = \lceil n/2 \rceil$, it is called the median problem.

**Example:** Given $\langle 1, 8, 23, 10, 19, 33, 100 \rangle$, the 4th smallest element is 19.

**Question:** How do you solve this problem?

## $\boxed{\textbf{First Solution: Selection by sorting}}$

**Step 1:** Sort the elements in ascending order with any algorithm of complexity $O(n \log n)$.

**Step 2:** Return the $i$th element of the sorted array.

The complexity of this solution is $O(n \log n)$.

**Question:** Can we do better?

**Answer:** YES, but we need to recall Partition($A, p, r$) used in Quicksort!

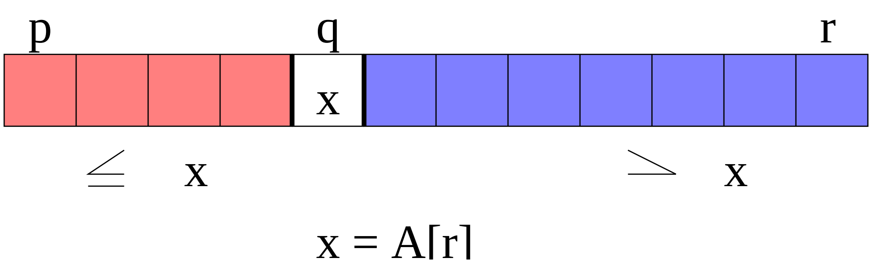
**Second Solution : Linear running time in average**

Recall of Partition($A, p, r$)

**Definition:** Rearrange the array $A[p..r]$ into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that

$$A[u] \leq A[q] < A[v]$$

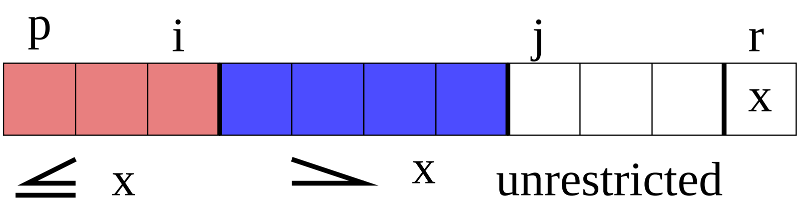
for any $p \leq u \leq q-1$ and $q+1 \leq v \leq r$.




(1) The original $A[r]$ is used as the pivot.
(2) It is a deterministic algorithm.
(3) The element for the $q$th position is found!

Note that this partition is different from the partition we used in COMP 171.

## The Idea of Partition($A, p, r$)

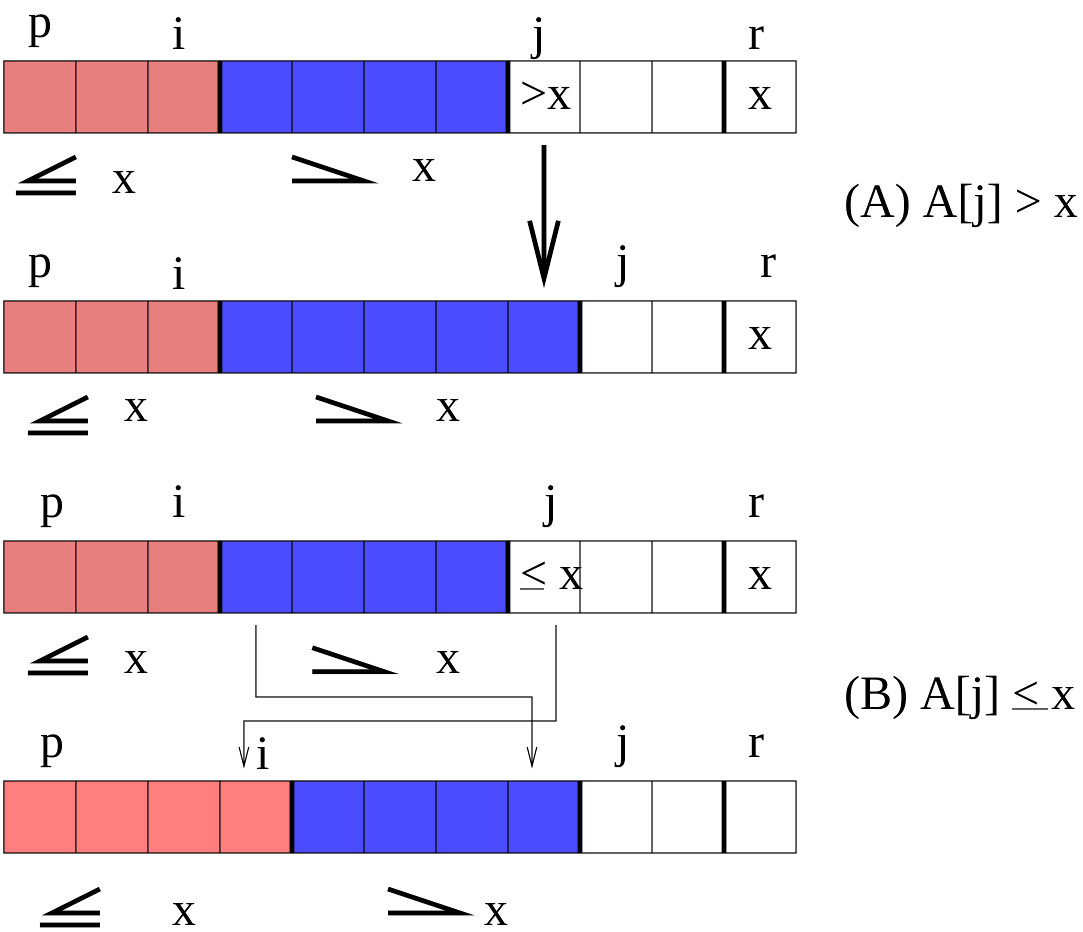



**(1)** Initially $(i, j) = (p - 1, p)$.

**(2)** Increase $j$ by 1 each time to find a place for $A[j]$.
At the same time increase $i$ when necessary.

**(3)** The procedure stops when $j = r$.

# One Iteration of the Procedure Partition




**(A)** Only increase $j$ by 1.

**(B)** $i \leftarrow i + 1$. $A[i] \leftrightarrow A[j]$. $j \leftarrow j + 1$.

## The Operation of Partition($A, p, r$): Example

| i | p, j | | | | | | | r | |
|---|---|---|---|---|---|---|---|---|---|
| | 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 | (1) |

| p, i | j | | | | | | r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 | (2) |

| p, i | | j | | | | | r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 | (3) |

| p, i | | | j | | | | r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4 | (4) |

| p | i | | | j | | | r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 7 | 8 | 3 | 5 | 6 | 4 | (5) |

| p | | i | | | j | | r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4 | (6) |

| p | | i | | | | j | r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4 | (7) |

| p | | i | | | | | j, r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4 | (8) |

| p | | i | | | | | j, r | |
|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 3 | 4 | 7 | 5 | 6 | 8 | (9) |

## The Partition($A, p, r$) Algorithm

```
Partition(A, p, r)
{
 // A[r] is the pivot element
 x = A[r];
 i = p-1;
 for (j = p to r-1) {
  if (A[j] <= x) {
   i = i+1;
   exchange A[i] and A[j]
  }
 }

 // put pivot in position
 exchange A[i+1] and A[r]
 // q = i+1
 return i+1;
}
```

## The Running Time of Partition($A, p, r$)

comparison of array elements

assignment, addition, comparison of loop variables

**Partition($A, p, r$):**

| | |
|---|---|
| $x = A[r]$ | 1 |
| $i = p - 1$ | 1 |
| **for** $j = p$ to $r - 1$ | $2(r - p)$ |
|     **if** $A[j] \leq x$ | $(r - p)$ |
|         $i = i + 1$ | $\leq (r - p)$ |
|         **exchange** $A[i] \leftrightarrow A[j]$ | $\leq 3(r - p)$ |
| **exchange** $A[i + 1] \leftrightarrow A[r]$ | 3 |
| **return** $i + 1$ | 1 |

Total: $(r - p)$ and $\leq \{6(r - p) + 6\}$

Running time is $\Theta(r - p)$ , that is, linear in the length of the array $A[p..r]$.

## **Randomized-Partition($A, p, r$)**

**The Idea:** In the algorithm Partition($A, p, r$), $A[r]$ is always used as the pivot $x$ to partition the array $A[p..r]$.

In the algorithm Randomized-Partition($A, p, r$), we randomly choose an $j$, $p \leq j \leq r$, and use $A[j]$ as pivot.

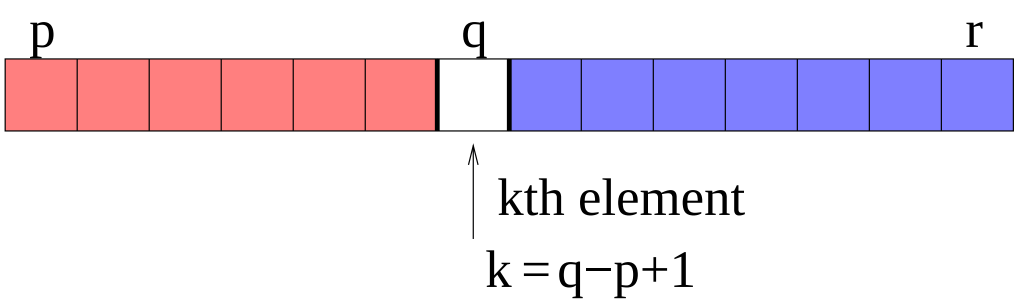
```
Randomized-Partition(A, p, r)
{
 j = random(p, r);
 exchange A[r] and A[j]
 Partition(A, p, r);
}
```

**Remark:** `random(`$p$`, `$r$`)` is a pseudorandom-number generator that returns a random number between $p$ and $r$.

## Randomized-Select($A, p, r, i$), $1 \leq i \leq r - p + 1$

**Problem:** Select the $i$th smallest element in $A[p..r]$, where $1 \leq i \leq r - p + 1$.

**Solution:** Apply Randomized-Partition($A, p, r$), getting




**Case 1:** $i = k$, pivot is the solution.

**Case 2:** $i < k$, the $i$th smallest element in $A[p..r]$ must be the $i$th smallest element in $A[p..q-1]$.

**Case 3:** $i > k$, the $i$th smallest element in $A[p..r]$ must be the $(i-k)$th smallest element in $A[q+1..r]$.

If necessary, recursively call the same procedure to the subarray.

**Randomized-Select($A, p, r, i$),** $1 \leq i \leq r - p + 1$

**if** $p == r$
  **return** $A[p]$
$q =$ Randomized-Partition$(A, p, r)$
$k = q - p + 1$
**if** $i == k$   the pivot is the answer
  **return** $A[q]$
**else if** $i < k$
  **return** Randomized-Select$(A, p, q - 1, i)$
**else**
  **return** Randomized-Select$(A, q + 1, r, i - k)$

**Remark:** To find the $i$th smallest element in $A[1..n]$, call Randomized-Select($A, 1, n, i$).

## $\boxed{\textbf{Running Time of Randomized-Select}(A, 1, n, i)}$

Let $T(n, i)$ be the average number of comparisons of array elements for $1 \le i \le n$.

Then $T(1,1) = 0$ and for $n > 1$ we get

$$\begin{aligned} T(n,i) = (n-1) & \qquad \text{initial partition} \\ +\tfrac{1}{n}\{\textstyle\sum_{k=1}^{i-1} T(n-k, i-k) & \qquad \text{recursion, } k < i \\ + \textstyle\sum_{k=i+1}^{n} T(k-1, i)\} & \qquad \text{recursion, } k > i \end{aligned}$$

We will prove by induction on $n$ that

$$T(n,i) < 4\,n$$

for all $n$ and $i$.

## $\boxed{\textbf{Proof that } T(n,i) < 4\,n}$

**Induction basis:** $T(1,1) = 0 < 4 \cdot 1$.
**Induction step:** Assume that $T(m,j) < 4\,m$ for all $m < n$ and $1 \le j \le m$. Then

$$
\begin{aligned}
& T(n,i) \\
= \;& n - 1 + \frac{1}{n}\{\sum_{k=1}^{i-1} T(n-k, i-k) + \sum_{k=i+1}^{n} T(k-1, i)\} \\
< \;& n - 1 + \frac{1}{n}\{\sum_{k=1}^{i-1} 4(n-k) + \sum_{k=i+1}^{n} 4(k-1)\} \\
= \;& n - 1 + \frac{1}{n}\{4n(i-1) - 4\frac{i(i-1)}{2} + 4\frac{n(n-1)}{2} - 4\frac{i(i-1)}{2}\} \\
= \;& n - 1 + \frac{1}{n}\{2n^2 - 6n + (4n+4)i - 4i^2\}.
\end{aligned}
$$

## Proof that $T(n,i) < 4\,n$

$$T(n,i) < n - 1 + \frac{1}{n} f(i),$$

where

$$f(x) = 2n^2 - 6n + (4n+4)x - 4x^2.$$

$$f'(x) = (4n+4) - 8x = 0$$

$$f''(x) = -8 < 0$$

for $x = (n+1)/2$. Hence

$$f(x) \le f((n+1)/2) = 3n^2 - 4n + 1$$

for all $x$. Therefore

$$T(n,i) \le n - 1 + 3n - 4 + \frac{1}{n} < 4n.$$

## $\boxed{\textbf{Running Time of Randomized-Select(}A, 1, n, i\textbf{)}}$

We proved that $T(n,i) < 4n$. Since $T(n,i) \geq n-1$, we have in particular that

$$T(n,i) = \Theta(n).$$

## Randomized-Quicksort Algorithm

We make use of the Randomized-Partition idea to develop a new version of quicksort.

```
Randomized-Quicksort(A, p, r)
{
 if (p < r) {
  q = Randomized-Partition(A, p, r);
  Randomized-Quicksort(A, p, q-1);
  Randomized-Quicksort(A, q+1, r);
 }
}
```

Does it run faster than the original version of quicksort?

## Running Time of the Randomized-Quicksort

Results :

**Worst Case:** $T(n) = \Theta(n^2)$.
**Average Case:** $T(n) = O(n \log n)$.

Clearly, the worst case is still $\Theta(n^2)$, what about the average case?

## Average running time of Randomized-Quicksort

Key observations:

- The running time of (randomized) quicksort is dominated by the time spent in (randomized) partition. In the partition procedure, the time is dominated by the *number of key comparisons*.

- When a pivot is selected, the pivot is compared with every other elements, then the elements are partitioned into two parts accordingly.

- Elements in different partition are NEVER compared with each other in *all* operations.

Tricks: We find the *expected* number of comparisons for **all** randomized-partition calls.

## Average running time of Randomized-Quicksort

Let $A$ be the input array which is a permutation of the $n$ distinct elements $z_1 < z_2 < \ldots < z_n$.

Let $X$ be the total number of comparisons performed in ALL calls to randomized-partition. Let $X_{ij}$ be the number of comparisons between $z_i$ and $z_j$, observe that $X_{ij}$ can only be 0 or 1. Our goal is to compute the expected value of $X$, i.e.,

$$\begin{aligned}
E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [Pr\{z_i \text{ is compared to } z_j\} \times 1 \\
&\quad + Pr\{z_i \text{ is not compared to } z_j\} \times 0] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr\{z_i \text{ is compared to } z_j\}
\end{aligned}$$

## Average running time of Randomized-Quicksort

It remains to show how to find $Pr\{z_i \text{ is compared to } z_j\}$.

For $1 \leq i \leq j \leq n$, let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ (remember $z_i < z_{i+1} < \ldots < z_j$).

Key observations:

- If $z_i$ or $z_j$ is selected as a pivot BEFORE any elements in $\{z_{i+1}, z_{i+2}, \ldots, z_{j-1}\}$, $z_i$ and $z_j$ will be compared.

- Conversely, if any element in $Z_{ij}$ other then $z_i$ or $z_j$ is selected as a pivot before $z_i$ and $z_j$, $z_i$ and $z_j$ will be placed in DIFFERENT partitions, and hence they will NOT compare with each other in ALL randomized-partition calls.

- ANY element other than the elements in $Z_{ij}$ has no effect to $Pr\{z_i \text{ is compared to } z_j\}$.

## Average running time of Randomized-Quicksort

It remains to find the probability that $z_i$ or $z_j$ is the first pivot chosen from $Z_{ij}$.

$$
\begin{aligned}
&Pr\{z_i \text{ is compared to } z_j\} \\
&\quad = Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad = Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} \\
&\qquad + Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&\quad = \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&\quad = \frac{2}{j-i+1}
\end{aligned}
$$

## Average running time of Randomized-Quicksort

Putting everything together, we have

$$
\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n)
\end{aligned}
$$

Hence, the expected number of comparisons is $O(n \lg n)$, which is the average running time of Randomized-Quicksort.