Selection

General problem: Given a list $L$ of $n$ keys, and $k \in [1..n]$, find $k$th smallest element in $L$

Special case: $k = \lceil n/2 \rceil$ ... the median

One solution: sort the keys into increasing order, return $k$th entry in the sorted list

$$k=8 \quad 5 \; 6 \; 2 \; 6 \; 1 \; 4 \; 1 \; 8 \; 4 \; 5 \; 5 \; 8 \; 3 \; 8 \; 9 \; 4$$

This takes time $O(n \log n)$

We can do better: linear time!

- a randomized algorithm with expected running time $O(n)$ (for arbitrary inputs)
  - but can take time $\Theta(n^2)$ with very small probability
- a deterministic algorithm with running time $O(n)$
Quick Select: a randomized selection algorithm

QSelect(L, k):

choose a “pivot” p from L at random
partition L into 3 sublists: $L_{<p}, L_{=p}, L_{>p}$

// this can be done using at most $n - 1$ comparisons,
// where $n := \text{length}(L)$

if $k \leq \text{length}(L_{<p})$ then
    return QSelect($L_{<p}, k$)

else if $k \leq \text{length}(L_{<p}) + \text{length}(L_{=p})$ then
    return $p$

else // $k > \text{length}(L_{<p}) + \text{length}(L_{=p})$
    return QSelect($L_{>p}, k - \text{length}(L_{<p}) - \text{length}(L_{=p})$)
Example

$k=8$

$p=3$

$k=4$

$p=8$

$k=4$

$p=5$
Intuition for running time:
we split the problem into two problems of “roughly equal” size (in linear time) and then solve one of them
reminds us of the recurrence $T(n) \leq T(n/2) + O(n)$
Master Theorem says: $T(n) = O(n)$
BUT . . . this is not rigorous: the splitting step is probabilistic, and we may get some “bad” splits

Potential quadratic time behavior
Suppose $k = n$
Suppose at every step, we get unlucky and choose as the pivot the smallest remaining element
We get a recurrence

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

which says $T(n) = \Theta(n^2)$
But this happens with very small probability
Let $C$ := number of comparisons

**Theorem.** $E[C] = O(n)$

For $j = 0, 1, 2, \ldots$ let $N_j$ := size of the subproblem at level $j$ (or zero if none)

**Claim.** $E[N_j] \leq (\frac{3}{4})^j n$ and for each $j = 0, 1, 2, \ldots$

**Using the claim:**

$$C \leq N_0 + N_1 + \cdots$$

$$E[C] \leq E[N_0] + E[N_1] + \cdots$$

$$\leq n \sum_{j \geq 0} (\frac{3}{4})^j$$

$$= n \cdot \frac{1}{1 - \frac{3}{4}} = 4n$$
Proof of Claim.

\( N_0 = n \)

Let’s first prove that \( \mathbf{E}[N_1] \leq \frac{3}{4}n \)

Imagine the keys in \( L \) are sorted

Let \( R \) the index of the pivot \( p \) in the sorted list

\( R \) is uniformly distributed over \( \{1, \ldots, n\} \)

\( \text{length}(L_{<p}) \leq R - 1 \) and \( \text{length}(L_{>p}) \leq n - R \)

\( \therefore N_1 \leq \max\{R - 1, n - R\} \)
A calculation . . .

Assume $R$ uniform over $\{1, \ldots, n\}$

Want to show: $E[\max\{R - 1, n - R\}] \leq \frac{3}{4} n$

NOTE: $E[\max\{X, Y\}] \leq \max\{E[X], E[Y]\}$

Proof by picture ($n = 8$):

Expectation $\leq 1/n$ times shaded area:

$$\leq \frac{1}{n} \times \frac{3}{4} n^2 = \frac{3}{4} n$$
To recap: we have proved $E[N_1] \leq \frac{3}{4} n$

What about $N_2$? **Use conditional expectation:**

$$E[N_2] = \sum_m E[N_2 | N_1 = m] \cdot \Pr[N_1 = m]$$

*same analysis as $N_1$*

$$\leq \sum_m \left( \frac{3}{4} m \right) \Pr[N_1 = m]$$

$$= \frac{3}{4} \sum_m m \Pr[N_1 = m]$$

$$= \frac{3}{4} E[N_1] \leq \left( \frac{3}{4} \right)^2 n$$

By induction: $E[N_j] \leq \left( \frac{3}{4} \right)^j n$ for $j = 0, 1, 2, \ldots$
Analysis of recursion depth

Let $D :=$ the depth of the recursion tree

**Theorem:** $E[D] = O(\log n)$

**Tail sum formula:** $E[D] = \sum_{j \geq 1} \Pr[D \geq j]

**Observe:** $D \geq j \iff N_j \geq 1$

**Markov says:** $\Pr[N_j \geq 1] \leq E[N_j] \leq (\frac{3}{4})^j n$
Set \( j_0 := \lfloor \log_{4/3} n \rfloor \) \quad // = \text{least } j \text{ such that } (\frac{3}{4})^j n \leq 1

We have:

\[
E[D] = \sum_{j \geq 1} \Pr[D \geq j]
\]

\[
= \sum_{j=1}^{j_0-1} \Pr[D \geq j] + \sum_{j=j_0}^{\infty} \Pr[D \geq j]
\]

\[
\leq (j_0 - 1) + \sum_{j=j_0}^{\infty} (\frac{3}{4})^j n
\]

\[
= (j_0 - 1) + ((\frac{3}{4})^{j_0} n) \sum_{j=j_0}^{\infty} (\frac{3}{4})^{j-j_0}
\]

\[
\leq \log_{4/3} n + 4
\]
Practical aspects: a fast, in-place 2-way partitioning algorithm

Assume keys are stored in an array $A[m..n)$

Assume keys are distinct and that the pivot $p$ is at $A[m]$ (if it’s initially at $A[k]$, then swap $A[m] \leftrightarrow A[k]$)

Invariant:

Two inner loops:

- moving $i$ to the right: skip over $<$, halt on $>$
- moving $j$ to the left: skip over $>$, halt on $<$

Swap $A[i++] \leftrightarrow A[j--]$

Repeat until $i$ crosses $j$

Swap $A[m] \leftrightarrow A[j]$
// 2-way partition of an array $A[m..n]$ about pivot $p = A[m]$

$i \leftarrow m + 1, j \leftarrow n - 1$

while $i \leq j$ do
    while $i < n$ and $A[i] < p$ do $i++$
    if $i \leq j$ then swap $A[i++] \leftrightarrow A[j--]$

swap $A[m] \leftrightarrow A[j]$

// First $j$ elements of $A$ are less than $p$
// $A[j] = p$
// Last $n - j - 1$ elements if $A$ are greater than $p$

**What if $A$ contains duplicates?**

When the above algorithm terminates, we have:

- First $j$ elements of $A$ are $\leq p$
- Last $n - j - 1$ elements if $A$ are $\geq p$

The above running time analysis no longer applies

if we change the comparisons $< / >$ to $\leq / \geq$, running time may degrade to *quadratic*

In both theory and practice, it is better to do true 3-way partitioning, which is not too hard to do
Practical aspects: a fast, in-place 3-way partitioning algorithm

An idea from Bentley & McIlroy, “Engineering a Sort Function” (1993)

Invariant:

\[
\begin{array}{ccccc}
\text{=} & < & ? & > & = \\
\text{a} & \text{b} & \text{c} & \text{d}
\end{array}
\]

Two inner loops:
- moving \( b \) to the right: skip over <, swap = w/ \( A[a++] \), halt on >
- moving \( c \) to the left: skip over >, swap = w/ \( A[d--] \), halt on <

Swap \( A[b++] \leftrightarrow A[c--] \)

Repeat until \( b \) crosses \( c \)

When finished, the \( = \)'s need to be swapped to the middle
// 3-way partition of an array \( A[m..n] \) about pivot \( p = A[m] \)
\[
a \leftarrow m + 1, \quad b \leftarrow m + 1, \quad c \leftarrow n - 1, \quad d \leftarrow n - 1
\]
while \( b \leq c \) do

\[
\text{while } b \leq c \text{ and } (cmp \leftarrow \text{compare}(A[b], p)) \leq 0 \text{ do}
\]
if \( cmp = 0 \) then swap \( A[a++] \) and \( A[b] \)
\( b++ \)

\[
\text{while } b < c \text{ and } (cmp \leftarrow \text{compare}(A[c], p)) \geq 0 \text{ do}
\]
if \( cmp = 0 \) then swap \( A[d--] \leftarrow A[c] \)
\( c-- \)

if \( b < c \) then swap \( A[b++] \leftarrow A[c--] \) 
else if \( b = c \) then \( c-- \) \hfill // must have \( \text{compare}(A[b], p) > 0 \)

// swap '='s on the left to the middle
\[
s \leftarrow \min(a, b - a)
\]
for \( i \) in \([0..s]\) do swap \( A[m + i] \leftarrow A[b - s + i] \)

// swap '='s on the right to the middle
\[
t \leftarrow \min(d - c, n - 1 - d)
\]
for \( i \) in \([0..t]\) do swap \( A[b + i] \leftarrow A[n - t + i] \)

// First \( b - a \) elements of \( A \) are less than \( p \)
// Last \( d - c \) elements if \( A \) are greater than \( p \)
// Number of comparisons \( \leq n - m + 1 \)
Example

\[ p = 5 \]

\[ \begin{array}{cccccccccccc}
5 & 1 & 6 & 6 & 5 & 4 & 1 & 8 & 4 & 5 & 5 & 8 & 3 & 4 & 9 & 8 \\
\end{array} \]

move

\[ \begin{array}{cccccccccccc}
5 & 1 & 6 & 6 & 5 & 4 & 1 & 8 & 4 & 5 & 5 & 8 & 3 & 4 & 9 & 8 \\
\end{array} \]

swap

\[ \begin{array}{cccccccccccc}
5 & 1 & 4 & 6 & 5 & 4 & 1 & 8 & 4 & 5 & 5 & 8 & 3 & 6 & 9 & 8 \\
\end{array} \]

move

\[ \begin{array}{cccccccccccc}
5 & 1 & 4 & 6 & 5 & 4 & 1 & 8 & 4 & 5 & 5 & 8 & 3 & 6 & 9 & 8 \\
\end{array} \]

swap

\[ \begin{array}{cccccccccccc}
5 & 1 & 4 & 3 & 5 & 4 & 1 & 8 & 4 & 5 & 5 & 8 & 6 & 6 & 9 & 8 \\
\end{array} \]

move

\[ \begin{array}{cccccccccccc}
5 & 5 & 4 & 3 & 1 & 4 & 1 & 8 & 4 & 9 & 8 & 8 & 6 & 6 & 5 & 5 \\
\end{array} \]

swap

\[ \begin{array}{cccccccccccc}
5 & 5 & 4 & 3 & 1 & 4 & 1 & 8 & 4 & 9 & 8 & 8 & 6 & 6 & 5 & 5 \\
\end{array} \]

swap to middle

\[ \begin{array}{cccccccccccc}
1 & 4 & 4 & 3 & 1 & 4 & 5 & 5 & 5 & 8 & 8 & 6 & 6 & 8 & 9 \\
\end{array} \]
Deterministic linear-time selection

Idea:

- divide $L$ into $\approx n/5$ blocks of size 5
- sort each block, and compute median of each block
- let $M :=$ the list of medians (so $\text{length}(M) \approx n/5$)
- recursively find the median $p$ of $M$
- use $p$ as the pivot, and proceed as in Quick Select
Consider a single recursive invocation

Local cost is $O(n)$

Both $\text{length}(L_{<p})$ and $\text{length}(L_{>p})$ are $\leq \frac{7}{10} n + O(1)$

Two recursive calls:

- one of size at most $\frac{1}{5} n + O(1)$
- one of size at most $\frac{7}{10} n + O(1)$
Sum of subproblem sizes $\leq 0.9n + c_1 + c_2$ for some constants $c_1$ and $c_2$

You worked a homework problem similar to this — the only difference was that $c_1 = c_2 = 0$ in the homework.

It’s not hard to generalize to allow non-zero $c_1$ and $c_2$.

$\therefore$ total cost is $O(n)$