THE SELECTION PROBLEM

• Input: An array $A$ containing $n$ distinct integer values, and an integer $k$ between 1 and $n$
• Output: The $k$th smallest integer in $A$
• Minimum is a special case where $k = 1$
• Median is a special case where $k = \frac{n}{2}$
• Maximum is a special case where $k = n$
• Simple algorithm for solving selection?

Recursive calls

Number of elements in this range $= l_y$

• What's the $k$th smallest element of $A$?
  • If $k = l_y$ then $y$
  • If $k < l_y$ then the $k$th smallest in $A_1$
  • If $k > l_y$ then the $(k - l_y)$th smallest in $A_R$

QuickSelect($k$, $A[l..r]$)

1. if $n = 1$ then return $A[1]$ // base case
3. if $k = 1$ then return $y$
4. if $k > 1$ then return QuickSelect($k - 1$, $AR$) // to avoid need for expansion
5. if $r = 1$ then return $y$
7. if $y < A[r]$ then $r = r - 1$ // $AR[0..r]$ is a new array
8. $AR[l] = a[l]$
9. $AR[r] = a[r]$
10. $m = (l + r) / 2$
11. $AR[m] = a[m]$
12. for $i := l$ to $r$ do
13. if $A[i] < y$ then $AR[m + 1] = A[i]$
14. else if $A[i] > y$ then $AR[m + 1] = A[i]$
15. return QuickSelect($k - 1$, $AR$) // $AR[0..r]$ is the new index of $y$
OVERLY OPTIMISTIC ANALYSIS

- If \( i_y = \frac{n}{2} \) then we recurse on \( \frac{n}{2} \) elements.
- We could always recurse on \( \frac{n}{2} \) elements.
  - We would get \( T(n) = T\left(\frac{n}{2}\right) + 8(n) \)
  - Which would yield \( a = 1, b = 2, y = 1, x = \log_2 1 = 0, y > x \text{ and } T(n) \in \Theta(n^3) = \Theta(n) \) by the Master theorem.

But we often end up recursing on \( \frac{n}{2} \) elements!

WORST-CASE ANALYSIS

- If we always get \( i_y = 1 \) and recurse on the right, then
  - We get \( T(n) = T(n-1) + \Theta(n) \)
  - By the substitution method this is \( \Theta(n^2) \)
- So, sometimes the pivot is good, sometimes it’s bad...
- What about the average case?

AVERAGE-CASE ANALYSIS

- Definition: we say a pivot is good if \( i_y \in \left(\frac{n}{4}, \frac{n}{2}\right) \)
- For any good pivot, we recurse on at most \( \frac{n}{2} \) elements.
- Probability of an arbitrary pivot being good?

Here is a more rigorous proof of the average-case complexity. We say the algorithm is in phase \( j \) if the current subarray has size \( x \), where

\[
\left(\frac{3}{4}\right)^{j+1} < x \leq \left(\frac{3}{4}\right)^{j}.
\]

Let \( X_j \) be a random variable that denotes the amount of computation time occurring in phase \( j \). If the pivot is in the middle half of the current subarray, then we transition from phase \( j \) to phase \( j + 1 \). This occurs with probability \( \frac{1}{2} \), so the expected number of recursive calls in phase \( j \) is \( 2 \). The computing time for each recursive call is linear in the size of the current subarray, so \( E[X_j] \leq 2n/4^j \) (where \( E[X] \) denotes the expectation of a random variable). The total time of the algorithm is given by \( X = \sum_{j \geq 1} X_j \). Therefore

\[
E[X] = \sum_{j \geq 1} E[X_j] \leq 2n \sum_{j \geq 1} 3/4^j = 8n \in O(n).
\]

TAKING SELECTION FURTHER

- We just showed:
  - QuickSelect with average-case runtime in \( \Theta(n) \)
- Next up:
  - Median-of-medians QuickSelect (MOQ)
  - Worst-case runtime in \( \Theta(n) \)

This is just for your notes, in case you want to know how you’d do this analysis formally.

The algorithm we will see picks a good pivot in every recursive call.
HIGH LEVEL ALGORITHM

- Similar to QuickSelect
- Choose a pivot
- Move smaller elements to the left of the pivot, and larger elements to the right of the pivot
- Recursively call MOMQuickSelect on one subarray
- Only difference is how we choose the pivot
- Always want to pick a good pivot

HOW GOOD IS THE PIVOT?

Recall median of each row

Imagine sorting each row

Then ordering row medians

# elements ≤ 23 is at least 3/10 of our 50-element input, or 3/18.

So, after sorting, rows must have at least 3/18 elements before and after

Is it a good pivot?

We recurse on $\delta_1$ or $\delta_2$, and both have size of mod 7

ALWAYS PICKING A GOOD PIVOT

Example input

A[1...50] =
11, 38, 6, 21, 20, 27, 4, 5, 7, 5, 8, 34, 49, 47, 28, 18, 44, 31, 44, 48, 27, 4, 5, 10, 30, 25, 14, 43, 22, 15, 32, 35, 41, 24, 1, 30, 22, 15, 26, 32, 36, 25, 37, 29, 45, 25, 49, 42

Time complexity for this step

Time complexity for this step

MOMQuickSelect(k, n, A)

// base case
if (n <= 14) then return(A[k])

// divide and conquer to find medians
r = \lceil \frac{n}{10} \rceil
medians([r])(k) = new array
for i = 1, 2, ..., r
    medians[i] = A([1, 2, ..., i])

sort(medians)

y = MOMQuickSelect(r, r)(medians)

// divide and conquer to find median of the r
\text{medians}([r])
\text{medians}([r])(k) = new array
for i = 1, 2, ..., r
    \text{sort}([\text{medians}([r]), \text{y}])

medians[r] = \text{y}

// divide and conquer to find rank k
AL, AR, y = Restructure(y, y)

if k <= y then return y
else if r < k then return MOMQuickSelect(k-y, AL)...
else if k > y then return MOMQuickSelect(k-y, AR)

MOMQuickSelect(k, 11, n = 14, 4)

// base case
if (n <= 14) then return(A[k])

// divide and conquer to find medians
r = \lceil \frac{n}{10} \rceil
medians([r])(k) = new array
for i = 1, 2, ..., r
    medians[i] = A([1, 2, ..., i])

sort(medians)

y = MOMQuickSelect(r, r)(medians)

// divide and conquer to find median of the r
\text{medians}([r])
\text{medians}([r])(k) = new array
for i = 1, 2, ..., r
    \text{sort}([\text{medians}([r]), \text{y}])

medians[r] = \text{y}

// divide and conquer to find rank k
AL, AR, y = Restructure(y, y)

if k <= y then return y
else if r < k then return MOMQuickSelect(k-y, AL)...
else if k > y then return MOMQuickSelect(k-y, AR)

MOMQuickSelect(k, 11, n = 21, A)

// base case
if (n <= 14) then return(A[k])

// divide and conquer to find medians
r = \lceil \frac{n}{10} \rceil
medians([r])(k) = new array
for i = 1, 2, ..., r
    medians[i] = A([1, 2, ..., i])

sort(medians)

y = MOMQuickSelect(r, r)(medians)

// divide and conquer to find median of the r
\text{medians}([r])
\text{medians}([r])(k) = new array
for i = 1, 2, ..., r
    \text{sort}([\text{medians}([r]), \text{y}])

medians[r] = \text{y}

// divide and conquer to find rank k
AL, AR, y = Restructure(y, y)

if k <= y then return y
else if r < k then return MOMQuickSelect(k-y, AL)...
else if k > y then return MOMQuickSelect(k-y, AR)
Time complexity?

```
[Diagram of array partitioning]
```

Want # ordered row operations

```
\[
\begin{array}{c|c|c|c}
0 & 1 & 2 & 3 \\
\hline
1 & 2 & 3 & 4 \\
\end{array}
\]
```

\( n \) iterations

```
\[
T(n) \in O(n) + T(n/5) + T(7n/10)
\]
```

\( \frac{T(n)}{n} \) grows to a maximum

```
\[
\frac{T(n)}{n} = C \frac{n}{10}
\]
```

\( T(n) \in O(n) \) if \( n \leq 14 \)

\( T(n) \in O(1) \) if \( n \geq 15 \)

\( T(n) \in O(n) \) if \( n \geq 15 \)

\( T(n) \in O(1) \) if \( n \leq 14 \)

```
\[
\sum \left( \frac{3}{10} \right)^i = 10 \approx 10(1)
\]
```

\( 7(n) \in O(n) + T(n/5) + T(7n/10) \) if \( n \geq 15 \)

\( 7(n) \in O(1) \) if \( n \leq 14 \)

```
\[
\text{Guess & check: } T(n) = cn
\]
```

\( T(n) = c'n + \left( \frac{3}{10} \right)^n \) where \( c' > 0 \)

Want to prove: \( T(n) = cn \) for some \( c > 0 \)

\( c' \) comes from the work of each level of recursion being \( 0(n) \)

\( c \) is a positive constant we are trying to show exists

I.H.: Suppose \( 3c > 0 \) \( T(n') = cn' \) for \( 15 \leq n' < n \)

\( T(n) = c'n + c \frac{3^n}{10} \) (by inductive hypoth.)

\( T(n) = cn \) (want this to be true)

\( c'n + c \frac{3^n}{10} = cn \) (equivalently)

\( c'n + c \frac{3^n}{10} = c \Leftrightarrow c = 10c' \) (by algebra)

How much does the problem shrink?

\( \bullet \) Shrink by at least \( 3(r + 1) \)

\( \bullet \) Problem size \( \equiv n = 10r + 5 \)

\( \bullet \) Subproblem size \( \leq n - \text{Shrink} = n - 3(r + 1) \)

\( \bullet \) \( = 10r + 5 - 3r - 3 = 7r + 2 \)

\( \bullet \) Express in terms of \( n \) using \( r = \frac{n - 5}{10} \)

\( \bullet \) Subproblem size \( \leq 7 \left( \frac{n - 5}{10} \right) + 2 \leq \frac{7n + 5}{10} \)

\( \bullet \) \( = \frac{7n}{10} - \frac{35}{10} + 2 = \frac{n}{10} + \frac{7}{10} \)

The closest pair problem

\( \text{When someone near you coughs} \)

\( \text{Hope you are not anti-vaxxer} \)

\( \text{Classroom} \)
THE CLOSEST PAIR PROBLEM

◆ Input: Set P of n 2D points
◆ Output: pair p and q s.t. dist(p, q) minimum over all pairs
◆ Break ties arbitrarily
◆ dist(p,q) = \((p.x - q.x)^2 + (p.y - q.y)^2\)

Can we Divide & Conquer?

◆ Like non-dominated points: sort by x-axis & divide in half

Claim that doesn’t require a proof: closest pair (p, q):
1. (p, q) both in L or
2. (p, q) both in R or
3. One of (p, q) in L and one of (p, q) in R

Observation 1

◆ Let \(\delta = \min(\text{dist(pair}_L), \text{dist(pair}_R))\)
◆ Then pair, (if closest globally) lies in the above 2\(\delta\)-wide green strip

Example for Observation 1

Q: Can p be part of a globally closest pair? A: No. Everything in R has dist > \(\delta\) to p.
And we already have a solution with dist = \(\delta\).

Observation 2

◆ Say, p (the lowest y valued point in strip) is in pair
◆ Then the other point can only lie in this \(\delta x \delta\) square.
Core Idea For Finding Spanning Pair
1. Start from lowest y valued point in the strip
2. Search the $\delta \times \delta$ square points on the opposite side
3. Repeat 1 & 2 for the next lowest y-valued point
4. So on and so forth...

A More Practical Idea
◆ Don’t differentiate between same and opposite side
◆ Just search the $2\delta \times \delta$ above rectangle each time

Switching sides might complicate code... Turns out it’s not needed to get good time complexity.
A More Practical Idea

- Don’t differentiate between same and opposite side
- Just search the $2\delta \times \delta$ above rectangle each time

Claim: loop performs $O(1)$ iterations!

Time complexity?

For a particular $i$, how many $j$ iterations occur?

Obs: as many as there are points in the $2\delta \times \delta$ rectangle.

Q: How many points can be in a $2\delta \times \delta$ rectangle?
A: As many as in the left $\delta \times \delta$ square + right $\delta \times \delta$ square.
So, $\delta$ is the smallest distance between any pair of points in this square! A point in the middle would rule out any other points. So, most efficient packing of points puts one in each corner (4 total).

**Observe:** as many as there are points in the $2\delta \times \delta$ rectangle.

Q: How many points can be in a $2\delta \times \delta$ rectangle? 
A: As many as in the left $\delta \times \delta$ square + right $\delta \times \delta$ square.

**Time complexity:**

For a particular $i$, how many $j$ iterations occur? 

For $i = 1..\text{len}(S)$, 
for $j = (i+1)\ldots\text{len}(S)$, 
if $S(i) \neq S(j)$ then break.

Can only contain eight points!

Points in $S$.

Loop performs at most eight iterations.
Each does $O(1)$ work, so entire loop does $O(1)$ work.
So, `findMinSpanningPair` does $O(n \log n)$ work.

**Time complexity:**

`closestPair(P, len(P))` sort(P) by x values
Recurse(P) $O(n \log n)$

`findMinSpanningPair(P, 2)` $O(n \log n)$

• Let $T(n)$ be runtime of `closestPair(P, len(P))`
• Let $T(n)$ be runtime of `findMinSpanningPair(P, 2)`

$T(n) = \Theta(n \log n)$
$T(n) = \Theta(n \log n)$
$T(n) = \Theta(n \log n)$
$\therefore T(n) = \Theta(n \log n)$
$\therefore T(n) = \Theta(n \log n)$
$\therefore T(n) = \Theta(n \log n)$

**Improving the previous algorithm**

• Sorting by y-values causes `findMinSpanningPair` to take $O(n \log n)$ time instead of $O(n)$ time.
• This happens in each recursive call, and dominates the running time.
• Avoid sorting $P$ over and over by creating another copy of $P$ that is pre-sorted by y-values.
This selection step preserves the y-sort order

Observe $P_x L$ and $P_y L$ contain the same points
(specifically the points with $x \leq x_{mid}$).

Moreover $P_x L$ is sorted by $x$ while $P_y L$ is sorted by $y$
And similarly for $P_x R$, $P_y R$.
No need to sort in Recurse!

Time complexity

$T(n) = \Theta(n \log n)$

Merge sort recurrence.
$T(n) \in \Theta(n \log n)$

So runtime for Shamos algorithm is in $\Theta(n \log n)$.