THE SELECTION PROBLEM

- Input: An array A containing n distinct integer values, and an integer k between 1 and n.
- Output: The k\textsuperscript{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?

Restructure (A, y)

12 4 6 27 23 17 40

Recursive calls

A after

Restructure (A, y)

Number of elements in this range = \ell_y

- What's the k\textsuperscript{th} smallest element of A?
  - If k = \ell_y then y
  - If k < \ell_y then the k\textsuperscript{th} smallest in A_1
  - If k > \ell_y then the (k - \ell_y)\textsuperscript{th} smallest in A_R

QuickSelect(k, A[0..n])

if n = 1 then return A[1] // base case

precondition: 1 \leq k \leq n

y = A[1] // pick an arbitrary pivot

if k = y return y

else if k < y then return QuickSelect(k, A_{\ell_y})

else // k > y

return QuickSelect(k - y + 1, A_{\ell_y})
**OVERLY OPTIMISTIC ANALYSIS**

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

---

**WORST-CASE ANALYSIS**

- \( A \) after Restructure \((A, y)\):
  - 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: a pivot \( y \) is **good** if \( i_y \in \left(\frac{n}{4}, \frac{3n}{4}\right) \)
  - \( \frac{n}{4} \) elements
  - \( \frac{n}{2} \) elements
  - \( \frac{3n}{4} \) elements

- For any good pivot, we recurse on at most \( \frac{n}{4} \) 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

\[
\frac{1}{4} < x \leq \frac{3}{4} \left(\frac{2}{3}\right)^{j+1}.
\]

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 \( \frac{1}{2} \). The computing time for each recursive call is linear in the size of the current subarray, so \( E[X_j] \leq 2n^\alpha(\frac{2}{3})^j \) (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

\[
E[X] = \sum_{j \geq 1} E[X_j] \leq 2n \sum_{j \geq 1} \left(\frac{2}{3}\right)^{2j} = 8n \in \Theta(n).
\]

**TAKING SELECTION FURTHER**

- We just showed:
  - QuickSelect with **average case** runtime in \( \Theta(n) \)
- Next up:
  - Median-of-medians QuickSelect (MEDianQuickSelect)
  - **worst case** runtime in \( \Omega(n) \)

The algorithm we will see picks a good pivot in every recursive call. Both the pivot is as good as possible.
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 ModifiedOMQuickSelect on one subarray
  • Only difference is how we choose the pivot
• Always want to pick a good pivot

Example input:
A[1...50]:

46, 48, 27, 4, 2, 50, 23, 45, 3, 13, 43, 22, 10, 32, 35, 41, 24, 11, 38, 6, 21, 20, 17, 5, 7, 9, 8, 34, 49, 41, 28, 18, 44, 31, 4, 48, 44, 25, 9, 5, 10, 23, 15, 43, 42, 16, 32, 15, 24, 1, 30, 12, 15, 26, 19, 36, 35, 37, 39, 25, 90, 46, 29, 42

ALWAYS PICKING A GOOD PIVOT

Example input:
A[1...10]:

5, 4, 3, 2, 1, 10, 9, 8, 7, 6

Time complexity for this step: \( \Theta(n) \)

Time complexity for this step: \( \Theta(n) \)

Recursive problem size: \( \frac{n}{2} \)

### How Good is the Pivot

- Selects pivot \( i \) so \( \frac{\pi}{2} \leq \frac{\pi}{2} \leq \frac{\pi}{2} \)
  - \( n \) is at least \( \frac{\pi}{2} \)
  - \( n \) is at least \( \frac{\pi}{2} \) of \( \frac{\pi}{2} \)
  - \( n \) is at least \( \frac{\pi}{2} \) of \( \frac{\pi}{2} \)
  - \( n \) is at least \( \frac{\pi}{2} \) of \( \frac{\pi}{2} \)

### Time Complexity

Example input:
A[1...50]:

Group into rows of 5

<table>
<thead>
<tr>
<th>11</th>
<th>38</th>
<th>6</th>
<th>21</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>34</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>44</td>
<td>31</td>
<td>44</td>
<td>48</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>50</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>32</td>
<td>35</td>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>19</td>
<td>36</td>
<td>33</td>
<td>37</td>
</tr>
<tr>
<td>17</td>
<td>25</td>
<td>10</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

Find median of each row

<table>
<thead>
<tr>
<th>11</th>
<th>38</th>
<th>6</th>
<th>21</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>34</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>13</td>
<td>44</td>
<td>31</td>
<td>44</td>
<td>48</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>50</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>32</td>
<td>35</td>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>19</td>
<td>36</td>
<td>33</td>
<td>37</td>
</tr>
<tr>
<td>17</td>
<td>25</td>
<td>10</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

Time complexity for this step: \( \Theta(n) \)

Time complexity for this step: \( \Theta(n) \)

Recursive problem size: \( \frac{n}{2} \)
Time complexity?

```
// base case
if n == 14 then sort(A) and return A[k]
// divide and conquer to find medians
r = floor(n/2)
if k <= r then return y
if k > r then return y
// divide and conquer to find ranks
(Al, A2, A3) = Restructure(A, y)
if k <= r then return RankQuickSelect(k, A1, B)
else if k > r + 1 then return RankQuickSelect(k, A3, B)
else return y
```

 HOW MUCH DOES THE PROBLEM SHRINK?

- Shrinks by at least $3(r + 1)$
- Problem size = $n = 10r + 5$
- Subproblem size $\leq n = 10r + 5 - 3r - 3 = 7r + 2$
- Express in terms of $n$ using $r = \left\lfloor \frac{n}{10} \right\rfloor$

$$
\text{Subproblem size } \leq T\left\lfloor \frac{n}{10} \right\rfloor + 7 \leq \frac{7n}{10} + 2
$$

$$
= \frac{7}{10} \cdot n + 2 = \frac{7}{10} \cdot n + 2 \leq \frac{7}{10} \cdot 10 + 2
$$

### Time complexity

```
// base case
if n == 14 then sort(A) and return A[k]
// divide and conquer to find medians
r = floor(n/2)
if k <= r then return y
if k > r then return y
// divide and conquer to find ranks
(Al, A2, A3) = Restructure(A, y)
if k <= r then return RankQuickSelect(k, A1, B)
else if k > r + 1 then return RankQuickSelect(k, A3, B)
else return y
```

LET $T(n) = c'n + \frac{T\left(\frac{n}{2}\right)}{10} + T\left(\frac{7n}{10}\right)$ WHERE $c' > 0$

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

- Note $c$ and $c'$ are independent constants
  - $c'$ comes from the work at each level of recursion being $O(n)$
  - $c$ is a positive constant we are trying to show exists
- I.H.: Suppose $3c > 0 : T(n') = cn'$ for $15 \leq n' < n$

\[
T(n) = c'n + \frac{T\left(\frac{n}{2}\right)}{10} + T\left(\frac{7n}{10}\right)
\]
(by inductive hypoth.)

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

\[
\Rightarrow c'n + \frac{T\left(\frac{n}{2}\right)}{10} + T\left(\frac{7n}{10}\right) = cn
\]
(equivalently)

\[
\Rightarrow c'n + \frac{c'n}{10} + T\left(\frac{7n}{10}\right) = cn
\]
(by algebra)

THE CLOSEST PAIR PROBLEM

- **LEWIN FAMILY'S NEIGHBORHOOD**
  - **DISTANCE BETWEEN PAIRS**
  - **PROBLEM STATEMENT**
  - **ALGORITHM**
  - **TIME COMPLEXITY**

- **GUESS & CHECK**
  - **INDUCTIVE HYPOTHESIS**
  - **BASE CASE**
  - **INDUCTIVE STEP**
  - **ASSESSMENT**

- **THEOREM**

- **PROOF**

- **CONCLUSION**

- **APPLICATION**

- **FURTHER RESEARCH**
**THE CLOSEST PAIR PROBLEM**

- **Input:** Set \( P \) of \( n \) 2D points
- **Output:** pair \( p \) and \( q \) s.t. \( \text{dist}(p, q) \) minimum over all pairs
- Break ties arbitrarily
- \( \text{dist}(p, q) = (p_x - q_x)^2 + (p_y - q_y)^2 \)

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

**Example for Observation 1**

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

**Observation 2**

- Say, \( p \) (the lowest \( y \)-valued point in strip) is in \( \text{pair}_s \)
  - Has to be on the opposite side & can't be \( > \delta \) higher than \( p \) on \( y \)-axis.
- Then the other point can only lie in this \( \delta \times \delta \) 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...

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...

Switching sides might complicate code...

A More Practical Idea

◆ Don’t differentiate between same and opposite side
◆ Just search the $2\delta \times \delta$ above rectangle each time
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!
**POINTS IN A $\delta \times \delta$ SQUARE**

- Recall $\delta$ is the smallest distance between any pair of points that are both in $L$ or both in $R$.
- Note this square is entirely in $L$ or entirely in $R$.

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).

---

**Time complexity**

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

---

**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.
Shamos’ algorithm (1975)

This selection step preserves the y-sort order

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

Moreover $P_x$ is sorted by $x$ while $P_y$ is sorted by $y$.

And similarly for $P_R$, $P_y$.

No need to sort in Recurse!

Total $\Theta(n)$ for this function

Time complexity

$T(n) = 2T(n/2) + O(n)$

Merge sort recurrence.

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

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