Analysis and Design of Algorithms

CS3230

Week 6

Order statistics

Diptarka Chakraborty

Ken Sung

Question 1

- A billionaire wants to buy a house for his own use.
- A property agent has n houses. He wants to sell a house to the billionaire.
- The property agent shows the billionaire n houses one by one. Whenever the billionaire likes the house more than the previous houses, he will pay a deposit to ensure he can buy this house.

- The property agent is smart. He knows the preference of the billionaire. He wants to maximize the number of deposit payments by the billionaire.

- What is the strategy of the property agent?
- How many rounds of deposit payments does the billionaire pay?

Answer

- The property agent shows the houses in the order of increasing preference score of the billionaire.
- For every round, since the billionaire sees a better house, he needs to pay deposit every round.
- In total, the billionaire needs to pay n rounds of deposit.

Question 2

- The billionaire is not stupid.
- Instead of following the sequence of houses proposed by the property agent, the billionaire randomly shuffles the ordering of the houses.
- With such a strategy, what is the expected number of rounds of deposits the billionaire need to pay?

Answer

- Let X_i be an indicator random variable that the *i*th visited house is better than all previous visited houses.
- $Pr(X_1=1) = 1$
- $Pr(X_2=1) = 1/2$ (since there are 2 ways to order two houses and there is one way that the 2^{nd} house is better)
- $Pr(X_i=1) = 1/i$.
- $E(X_i) = Pr(X_i=1) = 1/i$.

- The expected number of rounds of deposit payments is $E(X_1 + X_2 + \ldots + X_n) = E(X_1) + E(X_2) + \ldots + E(X_n) = 1/1+1/2+1/3+\ldots+1/n = \Theta(\ln n)$.

Admin

- Prog1 is available in codecrunch
  - Due day is 5 Mar (Week 7)

- Mid-term test:
  - Time: 7 Mar 2020 (Sat) 11:00am – 01:00pm.
  - Venue: To be determined
  - Open book

Admin

Consultation hours:

- Ken Sung: Friday 2-3pm (COM2-02-06)
- Diptarka: Tuesday 11:00-12:00 (COM2-03-17)
- Wei Liang (ganweiliang@u.nus.edu): Monday 13:00-1400 (AS6-04-01)
- Eldon Chung (eldon.chung@u.nus.edu): Wednesday 14:00-15:00 (COM1-B1-12)
- Govind Venugopalan (gv94@u.nus.edu): Wednesday 16:00-17:00 (COM1-01-20 Programming Lab 6)
- Tran Tan Phat (e0196695@u.nus.edu): Wednesday 16:00-17:00 (COM1 02-15)
- Joshua Casey Darian (joshuac@comp.nus.edu.sg): Thursday 13:00-14:00 (COM1 #01-18 MR5)
- Le Quang Tuan (e0313526@u.nus.edu): Monday 14:00-15:00 (COM1 02-15)
- Zhang Dongping (e0427788@u.nus.edu)

Algorithms commonly used as subroutines for solving problems

- Binary Search
- Sorting
- Select

Often overlooked

Order statistics

What is order statistics?

- Given an unsorted list of n elements, the ith order statistics is the ith smallest element (the element is also called the ***rank** i* element).
  - $i = 1$: ***minimum***;
  - $i = n$: ***maximum***;
  - $i = \lfloor (n+1)/2 \rfloor$ or $\lceil (n+1)/2 \rceil$: ***median***.

- Example:

6	10	13	5	8	3	2	11

  - i=1 (i.e. minimum): the value is 2
  - i=8 (i.e. maximum): the value is 13
  - i=4 (i.e. medium): the value is 6

How to find the ith smallest element?

- ***Naive algorithm***:
  1. Sort (say merge sort)
  2. Report the ith smallest element.

- Worst-case running time $= \Theta(n \lg n) + \Theta(1)$

  $= \Theta(n \lg n)$.

- Can we do better than $\Theta(n \lg n)$ time?

Select the ith smallest element requires at least Ω(n)

- Select the ith smallest element requires $\geq$ n steps.

- Proof: By contrary, suppose we can find the ith smallest element using < n steps.
- Since we perform less than n steps, we can only see at most n-1 elements.
- Let a be the unseen element.
- We cannot determine if a is the ith smallest element or not.
- We arrive at contradiction.

Can we find minimum better than $\Theta(n \lg n)$ time?

- The answer is **YES**!

$$
\begin{array}{l}
\textsc{Find-Min}(A) \\
\quad m \leftarrow 1 \\
\quad \textbf{for } i = 2 \text{ to } n \textbf{ do} \\
\quad\quad \textbf{if } (A[i] < A[m]) \textbf{ then } m \leftarrow i \\
\quad \textbf{return } m
\end{array}
$$

- The above algorithm runs in $\Theta(n)$ time, which is optimal.

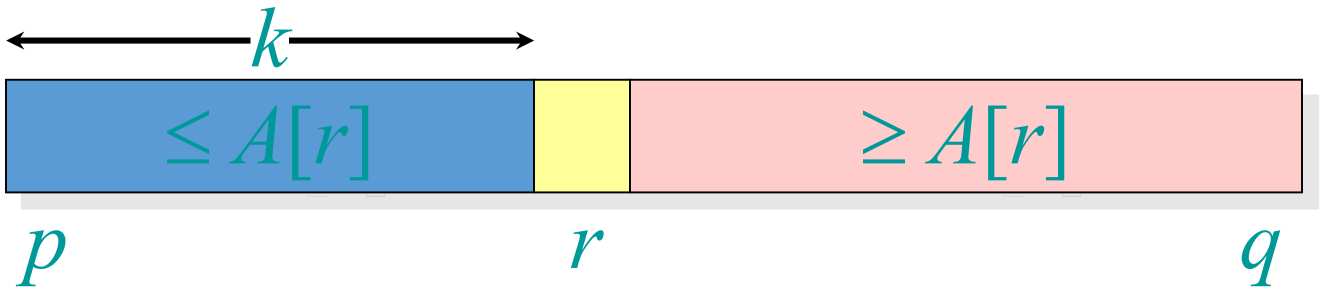
Can we find the rank-i element better than $\Theta(n \lg n)$ time?

- The answer is **YES**!

- We will present two solutions.
  - Randomized divide-and-conquer algorithm to select the rank-i element
  - Worst case linear time algorithm to select the rank-i element

Randomized divide-and-conquer algorithm

RAND-SELECT$(A[p..q], i)$ ⊳ ith smallest of $A[p..q]$
if $p = q$ **then return** $A[p]$
$r \leftarrow$ RAND-PARTITION$(A[p..q])$
$k \leftarrow r - p + 1$ ⊳ $k = \text{rank}(A[r])$
if $i = k$ **then return** $A[r]$
if $i < k$
  then return RAND-SELECT$(A[p..r-1], i)$
  else return RAND-SELECT$(A[r+1..q], i - k)$



Example

Select the $i = 7$th smallest:

6	10	13	5	8	3	2	11	$i = 7$

pivot

Partition:

2	5	3	6	8	13	10	11	$k = 4$

Select the $7 - 4 = 3$rd smallest recursively.

Question 3

For simplicity, you can assume that all elements are distinct.

- (Part 1) Suppose in every recursive call of Rand-Select, the size of the subarray is reduced from n to a size of at most 9n/10.
- What is the running time of Rand-Select?
- (Part 2) Suppose in every recursive call of Rand-Select, the size of the subarray is reduced from n to a size of at most n-1.
- What is the running time of Rand-Select?

- (a) $O(n)$ for Part 1 and $O(n)$ for Part2
- (b) $O(n^2)$ for Part 1 and $O(n)$ for Part 2
- (c) $O(n)$ for Part 1 and $O(n^2)$ for Part 2
- (d) $O(n^2)$ for Part 1 and $O(n^2)$ for Part 2

Answer

The answer is (c)

Part 1: Not unlucky

$$T(n) = T(9n/10) + \Theta(n) = \Theta(n)$$

$n^{\log_{10/9} 1} = n^0 = 1$

CASE 3

Part 2: Unlucky

$$T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$$

arithmetic series

Worse than sorting!

Analysis of expected time

The analysis follows that of randomized quicksort, but it's a little different.

Let $T(n)$ = the random variable for the running time of RAND-SELECT on an input of size n. Assume random numbers are independent.

For $k = 0, 1, \ldots, n-1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

Analysis (continued)

To obtain an upper bound, assume that the ith element always falls in the larger side of the partition:

When $X_k=1$,

$T(n) = T(\max\{k, n-k-1\}) + \Theta(n)$ for $k : n-k-1$ split

$$T(n) = \begin{cases} T(\max\{0, n-1\}) + \Theta(n) & \text{for } X_0=1, \\ T(\max\{1, n-2\}) + \Theta(n) & \text{for } X_1=1, \\ \quad \ldots & \\ T(\max\{n-1, 0\}) + \Theta(n) & \text{for } X_{n-1}=1. \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left(T(\max\{k, n-k-1\}) + \Theta(n) \right).$$

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k \big(T(\max\{k, n-k-1\}) + \Theta(n)\big)\right]$$

Take expectations of both sides.

Calculating expectation

$$E[T(n)] = E\left[\sum_{k=0}^{n-1} X_k\left(T(\max\{k, n-k-1\}) + \Theta(n)\right)\right]$$
$$= \sum_{k=0}^{n-1} E\left[X_k\left(T(\max\{k, n-k-1\}) + \Theta(n)\right)\right]$$

Linearity of expectation.

Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k\big(T(\max\{k, n-k-1\}) + \Theta(n)\big)\right] \\
&= \sum_{k=0}^{n-1} E\big[X_k\big(T(\max\{k, n-k-1\}) + \Theta(n)\big)\big] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)]
\end{aligned}
$$

Independence of X_k from other random choices.

Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k\big(T(\max\{k, n-k-1\}) + \Theta(n)\big)\right] \\
&= \sum_{k=0}^{n-1} E\big[X_k\big(T(\max\{k, n-k-1\}) + \Theta(n)\big)\big] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)] \\
&= \frac{1}{n}\sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\})] + \frac{1}{n}\sum_{k=0}^{n-1} \Theta(n)
\end{aligned}
$$

Linearity of expectation; $E[X_k] = 1/n$.

Calculating expectation

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \bigl(T(\max\{k, n-k-1\}) + \Theta(n)\bigr)\right] \\
&= \sum_{k=0}^{n-1} E\bigl[X_k \bigl(T(\max\{k, n-k-1\}) + \Theta(n)\bigr)\bigr] \\
&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)] \\
&= \frac{1}{n} \sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\})] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \\
&\le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)
\end{aligned}
$$

Upper terms appear twice.

Hairy recurrence

(But not quite as hairy as the quicksort one.)

$$E[T(n)] = \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)$$

Prove: $E[T(n)] \le cn$ for constant $c > 0$.

- The constant c can be chosen large enough so that $E[T(n)] \le cn$ for the base cases.

Use fact: $\displaystyle\sum_{k=\lfloor n/2 \rfloor}^{n-1} k \le \tfrac{3}{8}n^2$ (exercise).

Substitution method

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n)$$

Substitute inductive hypothesis.

Substitution method

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n)$$

$$\leq \frac{2c}{n}\left(\frac{3}{8}n^2\right) + \Theta(n)$$

Use fact.

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n) \\
&\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + \Theta(n) \\
&= cn - \left(\frac{cn}{4} - \Theta(n)\right)
\end{aligned}
$$

Express as ***desired*** – ***residual***.

Substitution method

$$
\begin{aligned}
E[T(n)] &\le \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} ck + \Theta(n) \\
&\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + \Theta(n) \\
&= cn - \left(\frac{cn}{4} - \Theta(n)\right) \\
&\le cn\,,
\end{aligned}
$$

if c is chosen large enough so that $cn/4$ dominates the $\Theta(n)$.

Question 4

Both randomized select and randomized quicksort are divide and conquer algorithms. Which of the following is true?

- Randomized select recurses on smaller subproblems.
- Randomized select recurses on fewer subproblems.
- Randomized select's runtime for the divide and combine step is asymptotically smaller.

Solution

- The answer is (B): Randomized select recurses on fewer subproblem.

- Both algorithms randomly select a pivot to partition the array.

- Randomized quicksort recurses on both partitions resulting in an expected runtime of $\Theta(n \lg n)$.

- Randomized quickselect recurses on only one of the subproblems resulting in an expected runtime of $\Theta(n)$.

Solution

- (A) is not correct since it may recurses on big subproblem.
- (C) is not correct since partition is still O(n).

Question 5

An adversary can construct an input of size n to force randomized quickselect to run in $\Omega(n^2)$ time.

- True
- False

Solution

- **False.** The adversary cannot construct an input that will run in $\Omega(n^2)$ time.
- In fact, for any input provided by the adversary, the randomized quickselect algorithm always runs in expected $\Theta(n)$ time.
- To force the randomized quickselect to run in $\Omega(n^2)$ time, we need to fix the randomization.
- Since the adversary cannot control the randomization, it has no way to force the algorithm to run in $\Omega(n^2)$ time.

Question 6

Who is the Master of Algorithms pictured below?

- Robert Tarjan
- Robert Floyd
- Richard Karp
- Tony Hoare

Robert Floyd

Turing Award winner

- One of the inventors of linear time select (Blum, Floyd, Pratt, Rivest, Tarjan)
- Known for Floyd-Warshall algorithm for all-pairs shortest path, Floyd-Hoare logic (use invariants for correctness)
- Did not have a PhD!

Summary of randomized order-statistic selection

- Works fast: linear expected time.
- Excellent algorithm in practice.
- But, the worst case is ***very*** bad: $\Theta(n^2)$.

Q. Is there an algorithm that runs in linear time in the worst case?

A. Yes, due to Blum, Floyd, Pratt, Rivest, and Tarjan [1973].

IDEA: Generate a good pivot recursively.

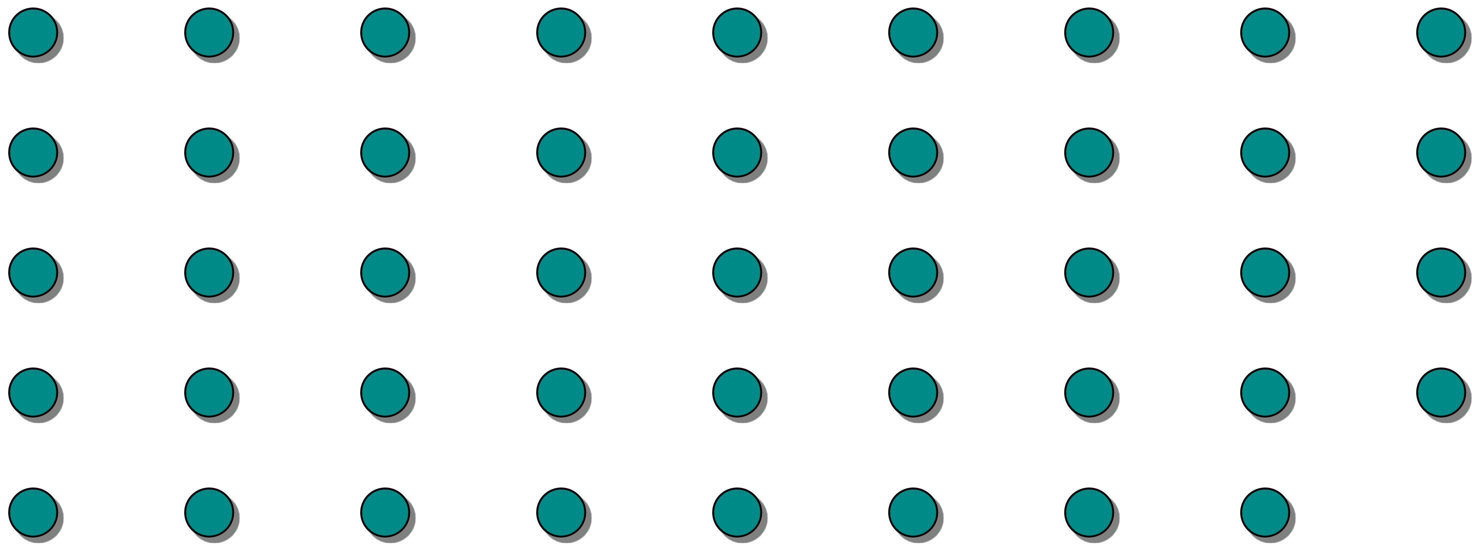
Worst-case linear-time order statistics

SELECT(i, n)

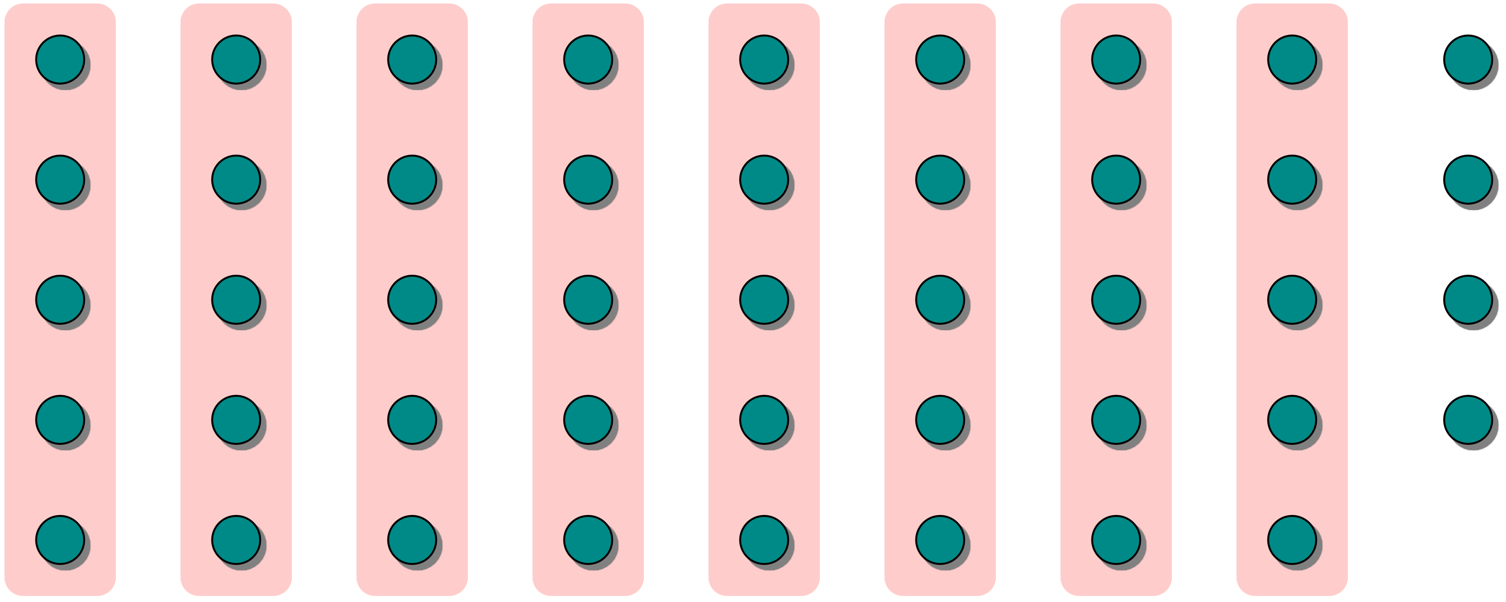
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
3. Partition around the pivot x. Let $k = \text{rank}(x)$.
4. **if** $i = k$ **then return** x
   elseif $i < k$
   then recursively SELECT the ith smallest element in the lower part
   else recursively SELECT the $(i–k)$th smallest element in the upper part

Same as RAND-SELECT

Choosing the pivot

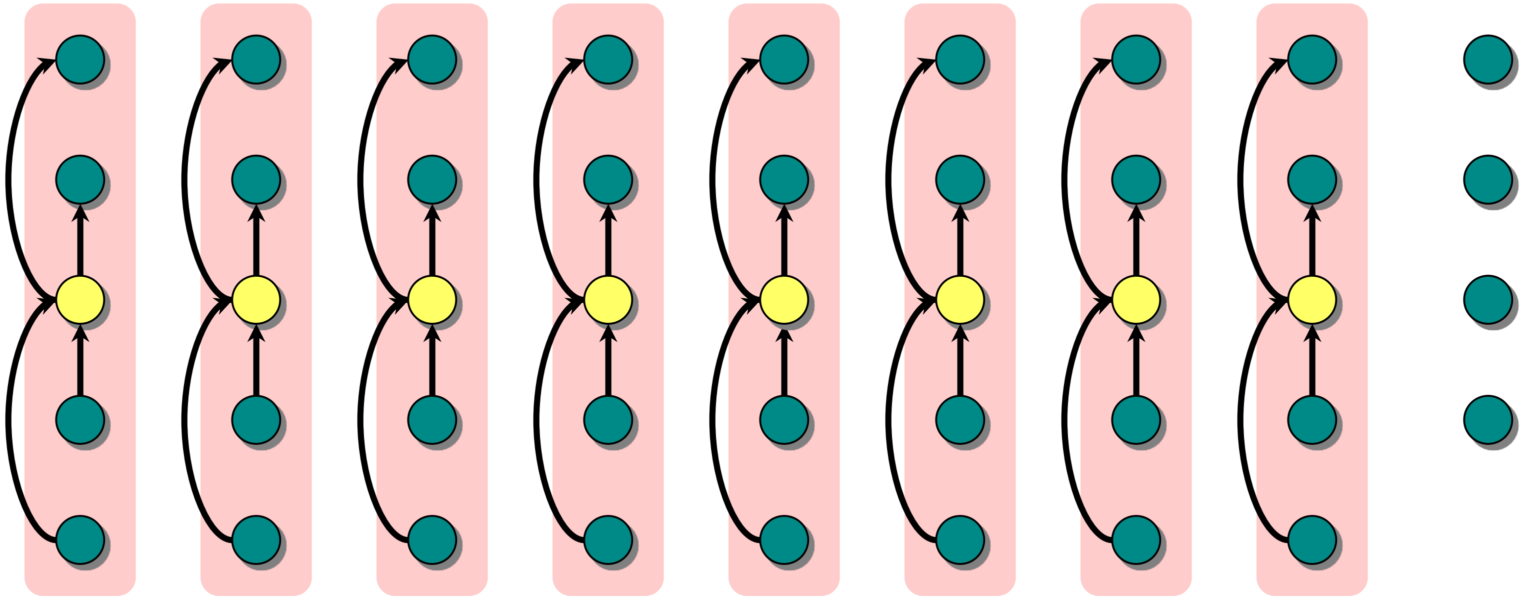


Choosing the pivot

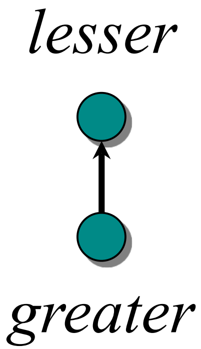


1. Divide the n elements into groups of 5.

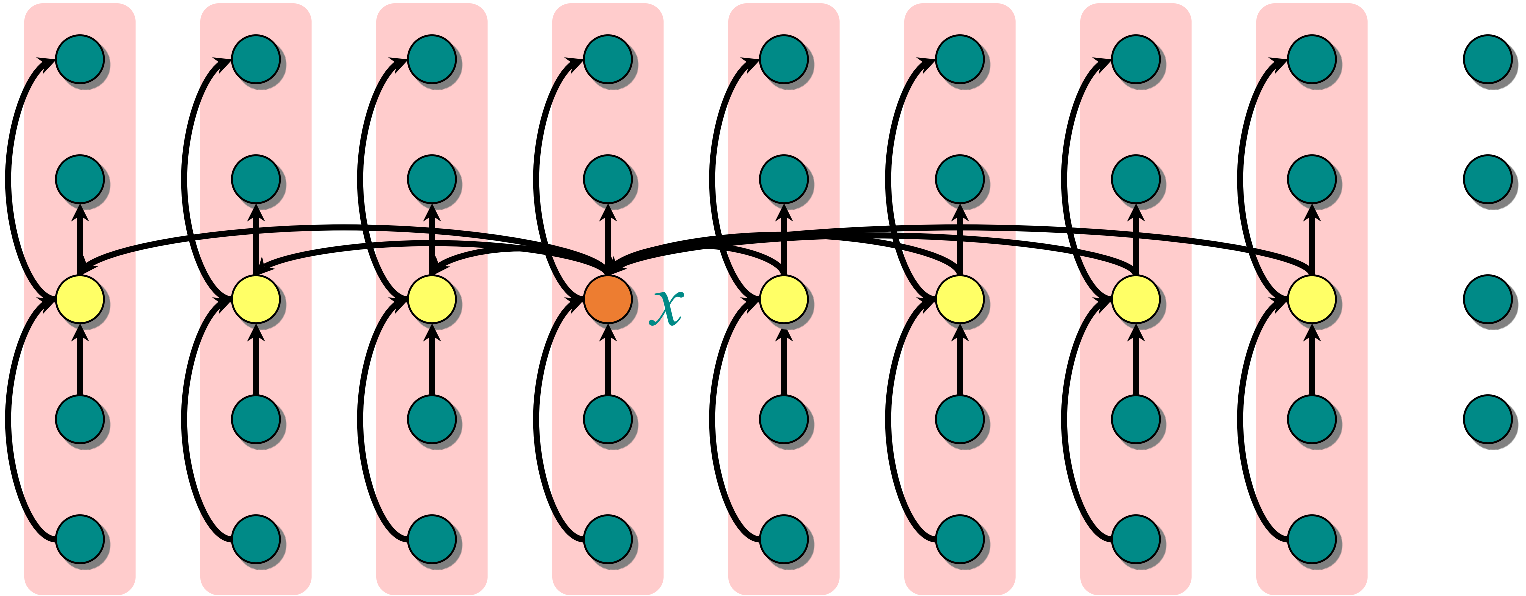
Choosing the pivot



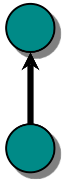
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.



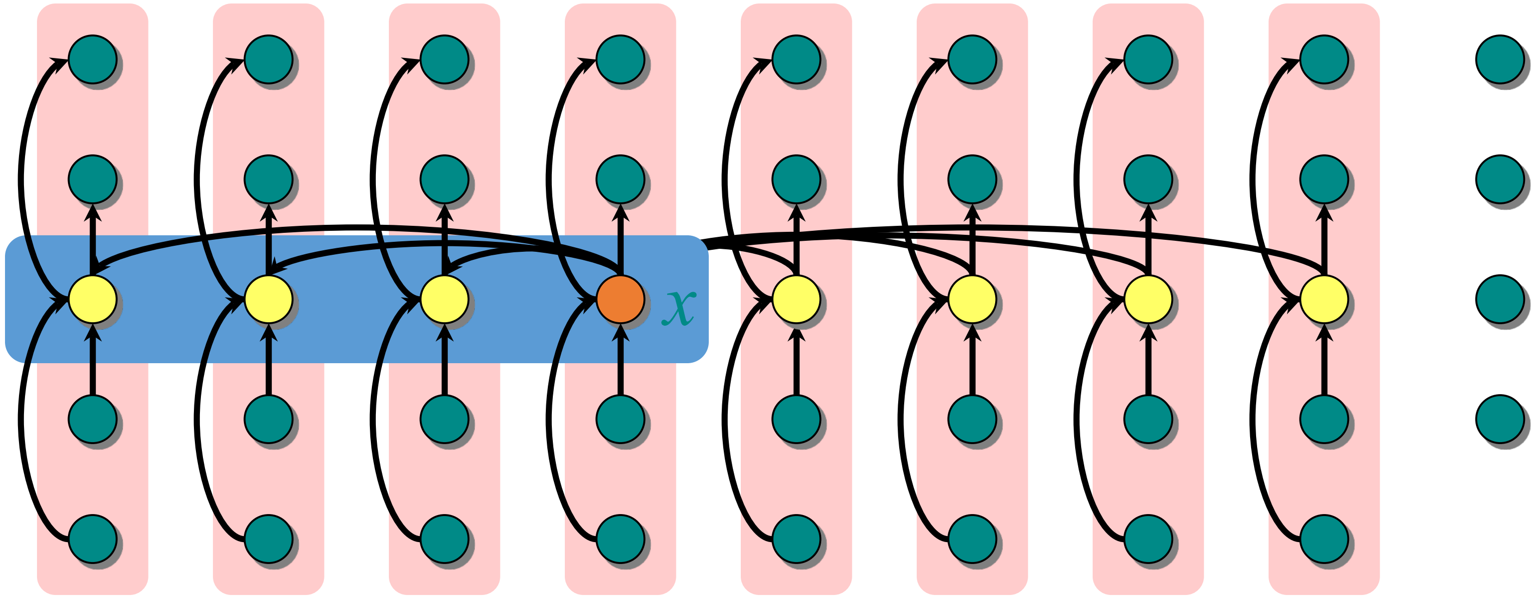
Choosing the pivot



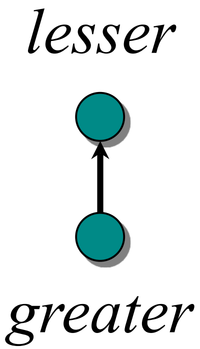
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.



Analysis

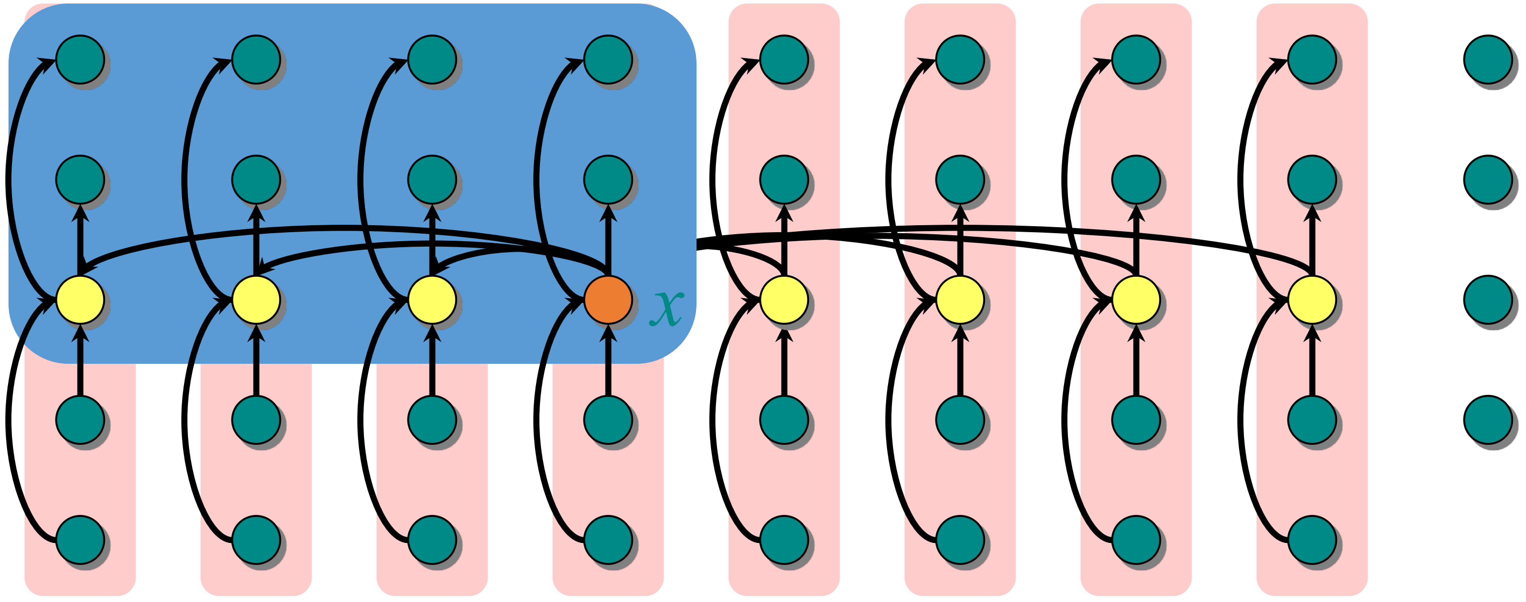


At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.



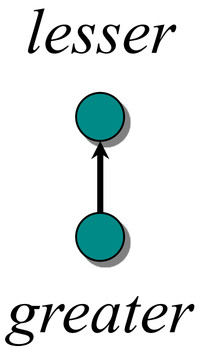
Analysis

(Assume all elements are distinct.)



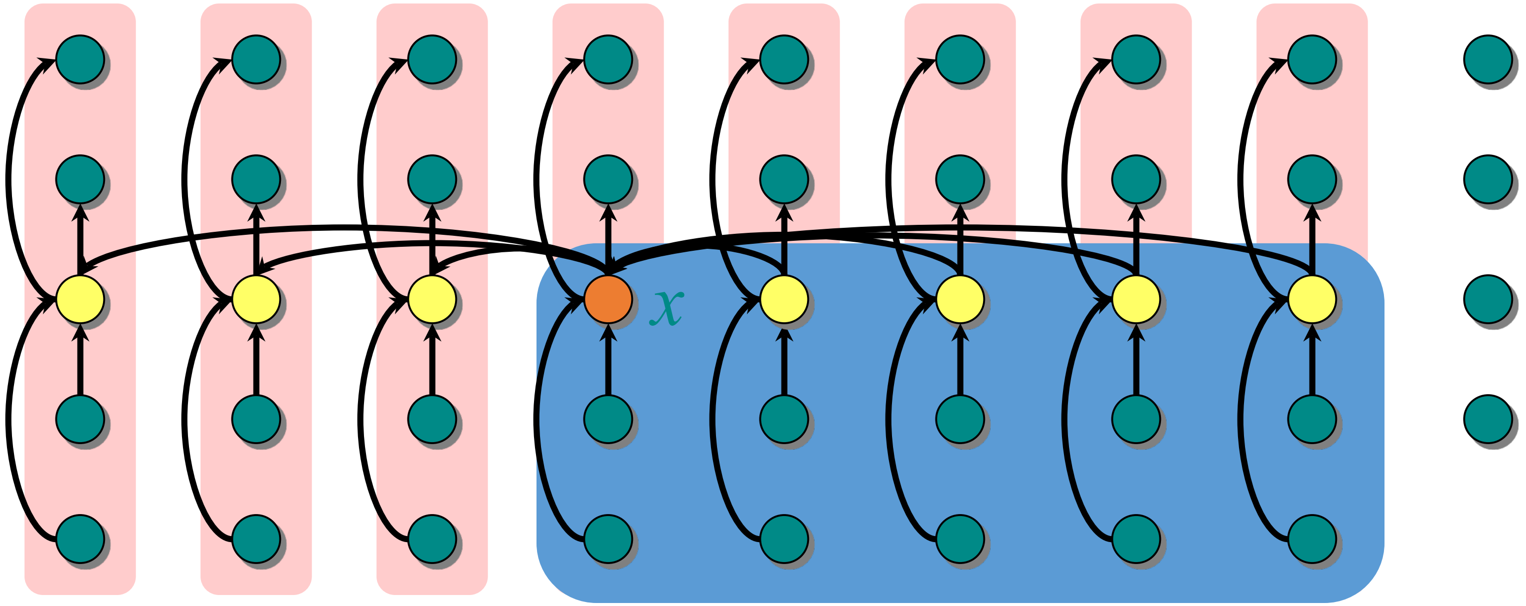
At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\leq x$.



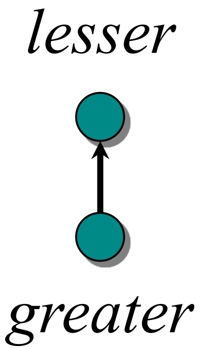
Analysis

(Assume all elements are distinct.)



At least half the group medians are $\leq x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\leq x$.
- Similarly, at least $3\lfloor n/10 \rfloor$ elements are $\geq x$.



Minor simplification

- For $n \geq 50$, we have $3\lfloor n/10 \rfloor \geq n/4$.
- Therefore, for $n \geq 50$ the recursive call to SELECT in Step 4 is executed recursively on $\leq 3n/4$ elements.
- Thus, the recurrence for running time can assume that Step 4 takes time $T(3n/4)$ in the worst case.
- For $n < 50$, we know that the worst-case time is $T(n) = \Theta(1)$.

Developing the recurrence

$T(n)$	SELECT(i, n)
$\Theta(n)$	1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
$T(n/5)$	2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
$\Theta(n)$	3. Partition around the pivot x. Let $k = \text{rank}(x)$.
	4. **if** $i = k$ **then return** x **elseif** $i < k$
$T(3n/4)$	**then** recursively SELECT the ith smallest element in the lower part **else** recursively SELECT the $(i–k)$th smallest element in the upper part

Solving the recurrence

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{3}{4}n\right) + \Theta(n)$$

Substitution:

$T(n) \le cn$

$$\begin{aligned} T(n) &\le \frac{1}{5}cn + \frac{3}{4}cn + \Theta(n) \\ &= \frac{19}{20}cn + \Theta(n) \\ &= cn - \left(\frac{1}{20}cn - \Theta(n)\right) \\ &\le cn \quad , \end{aligned}$$

if c is chosen large enough to handle both the $\Theta(n)$ and the initial conditions.

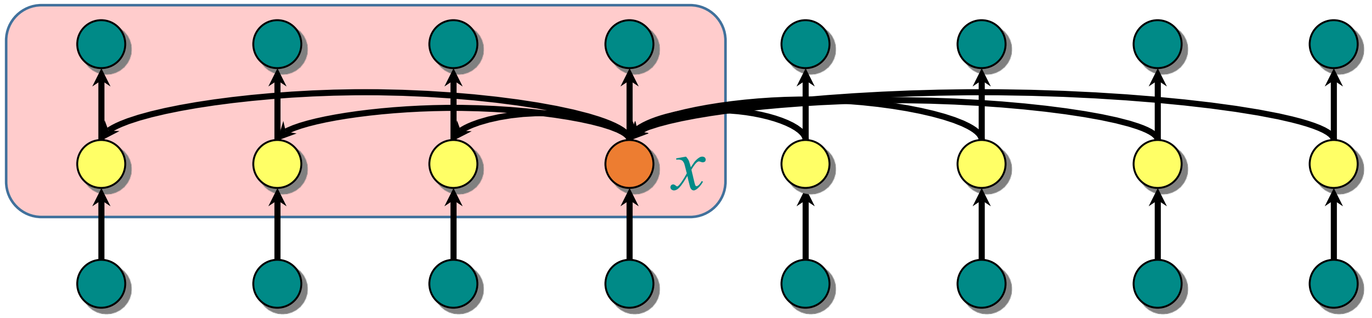
Conclusion

- The worst-case linear select runs in linear time.
- In practice, this algorithm runs slowly, because the constant in front of n is large.
- The randomized algorithm is far more practical.

Question 8

- The linear select algorithm divides the array into groups of 5.
- If we divide the array into groups of 3, will the algorithm work in linear time?

Solution



1. Divide the n elements into groups of 3. Find the median of each 3-element group by rote in $\Theta(n)$ time.
2. Recursively SELECT the median x of the $n/3$ group medians to be the pivot in $T(n/3)$ time.
   - At most $n/6$ group medians are $\leq x$.
   - Hence, the SELECT in Step 4 is executed recursively on at least $\mathrm{n} - 2\left(\frac{n}{6}\right) = \frac{2n}{3}$ elements.

4. Thus, the recurrence for running time for Step 4 takes at least $T(2n/3)$ time.

Solution

$T(n)$	SELECT(i, n)
$\Theta(n)$	1. Divide the n elements into groups of 3. Find the median of each 3-element group by rote.
$T(n/3)$	2. Recursively SELECT the median x of the $\lfloor n/3 \rfloor$ group medians to be the pivot.
$\Theta(n)$	3. Partition around the pivot x. Let $k = \text{rank}(x)$.
	4. **if** $i = k$ **then return** x **elseif** $i < k$
$\geq T(2n/3)$	**then** recursively SELECT the ith smallest element in the lower part **else** recursively SELECT the $(i–k)$th smallest element in the upper part

Solution

$$T(n) \geq T\left(\frac{1}{3}n\right) + T\left(\frac{2}{3}n\right) + \Theta(n)$$

- We can show that $T(n) = \Omega(n \lg n)$.
- Hence, this is not a linear time algorithm.

Question 7

- The linear select algorithm divides the array into groups of 5.
- If we divide the array into groups of 7, will the algorithm work in linear time?

Solution

SELECT(i, n)

1. Divide the n elements into groups of 7. Find the median of each 7-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/7 \rfloor$ group medians to be the pivot.
3. Partition around the pivot x. Let $k = \text{rank}(x)$.
4. **if** $i = k$ **then return** x
   elseif $i < k$
   then recursively SELECT the ith smallest element in the lower part
   else recursively SELECT the $(i–k)$th smallest element in the upper part

Solution

2. Recursively SELECT the median x of the $\lfloor n/7 \rfloor$ group medians to be the pivot. Hence, step 2 takes $T(n/7)$ time in the worst case.
   - Each group median is larger than or equal to 4 elements.
   - Therefore, at least $4\lfloor n/14 \rfloor$ elements are $\leq x$.
   - Similarly, at least $4\lfloor n/14 \rfloor$ elements are $\geq x$.
   - We have $4\lfloor n/14 \rfloor \geq n/4$.
4. Therefore, the recursive call to SELECT in Step 4 is executed recursively on $\leq 3n/4$ elements.
   - Thus, the recurrence for running time can assume that Step 4 takes time $T(3n/4)$ in the worst case.

Solution

$T(n)$	SELECT(i, n)
$\Theta(n)$	1. Divide the n elements into groups of 7. Find the median of each 7-element group by rote.
$T(n/7)$	2. Recursively SELECT the median x of the $\lfloor n/7 \rfloor$ group medians to be the pivot.
$\Theta(n)$	3. Partition around the pivot x. Let $k = \text{rank}(x)$.
	4. **if** $i = k$ **then return** x **elseif** $i < k$
$T(3n/4)$	**then** recursively SELECT the ith smallest element in the lower part **else** recursively SELECT the $(i–k)$th smallest element in the upper part

Solution

$$T(n) = T\left(\frac{1}{7}n\right) + T\left(\frac{3}{4}n\right) + \Theta(n)$$

- We can show that $T(n) = \Theta(n)$.
- Hence, this is a linear time algorithm.

Question 9

- You are a programmer in MOE (Ministry of Education). Your supervisor asks you to obtain a list of the top 100 PSLE score students in sorted order.
- Can you propose an efficient algorithm?
- What is the running time?

Solution

- Let n be the number of PSLE students.

- 1. Select the 100^{th} highest PSLE score student in $\Theta(n)$ time
- 2. Run partition to extract the top 100 PSLE score students in $\Theta(n)$ time
- 3. Sort the top 100 PSLE students by their score in $\Theta(1)$ time

- Total time is $\Theta(n)$.

Question 10

Given a set of points $p_1, \ldots, p_n$ on the real line, give a fast algorithm to find a point x that minimizes

(For convenience, you may assume that n is odd.)

$$\sum_{i=1}^{n} |p_i - x|$$

Solution

Lemma: Let x be the median of $p_1, \ldots, p_n$. $\mathrm{f(y)} \geq f(x)$ for any y.

$$\mathrm{f(y)} = \sum_{i=1}^{n} |p_i - y|$$

Proof: We argue that for any $y \neq x$,

$$f(y) - f(x) = \sum_{i=1}^{n} (|p_i - y| - |p_i - x|) \geq 0$$

$|p_i - y| - |p_i - x| = y - x$
for $p_i < x$.

$|p_i - y| - |p_i - x| \geq x - y$
for $x \leq p_i \leq y$
The value is minimized when
$p_i = y$, giving value $x - y$.

$|p_i - y| - |p_i - x| = x - y$
for $p_i > y$.

Case 1: $y \geq x$.

For points $p_i \leq x$, $|p_i - y| - |p_i - x| = (y - x)$.
For points $p_i \geq x$, $|p_i - y| - |p_i - x| \geq (x - y)$.

$$\sum_{i=1}^{n} |p_i - y| - |p_i - x|$$

$$\geq \sum_{i=1}^{\left\lfloor \frac{n}{2} \right\rfloor} (y - x) + \sum_{i=\lceil n/2 \rceil + 1}^{n} (x - y)$$

$$= 0$$

(WLOG, assume p_i are sorted.)

Case 2: $y \leq x$. This case can be showed similarly.

Solution: Run select to get the median x in linear time.