Data Structures and Algorithms

dongmassimo@gmail.com

October 26, 2019

- Questions (Tu Cao)

Ex. (8.6 Lower bound on merging sorted lists (c)(d))

- *Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.*
- *Show a lower bound of $2n - 1$ comparsions for merging two sorted lists.*

Ex. (8.6 Lower bound on merging sorted lists (c)(d))

- *Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.*
- *Show a lower bound of $2n-1$ comparsions for merging two sorted lists.*

- Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

- Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

Proof.

$$A = (a_1, \ldots, a_n), B = (b_1, \ldots, b_n) \to (\ldots, a_s, b_t, \ldots)$$

- Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

Proof.

$$A = (a_1, \dots, a_n), B = (b_1, \dots, b_n) \to (\dots, a_s, b_t, \dots)$$

goal: $\forall$correct algorithm $\mathcal{P}, a_s$ and b_t are compared in $\mathcal{P}$

- Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

Proof.

$$A = (a_1, \dots, a_n), B = (b_1, \dots, b_n) \to (\dots, a_s, b_t, \dots)$$

goal: $\forall$correct algorithm $\mathcal{P}, a_s$ and b_t are compared in $\mathcal{P}$

$$\forall \mathcal{P}, (\mathcal{P} \text{ is correct }) \implies (a_s \text{ and } b_t \text{ are compared})$$

- Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

Proof.

$$A = (a_1, \dots, a_n), B = (b_1, \dots, b_n) \to (\dots, a_s, b_t, \dots)$$

goal: $\forall$correct algorithm $\mathcal{P}, a_s$ and b_t are compared in $\mathcal{P}$

$$\forall \mathcal{P}, (\mathcal{P} \text{ is correct }) \implies (a_s \text{ and } b_t \text{ are compared})$$

$$\forall \mathcal{P}, (a_s \text{ and } b_t \text{ are } \textbf{not} \text{ compared}) \implies (\mathcal{P} \text{ is incorrect})$$

- Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

Proof.

$$A = (a_1, \ldots, a_n), B = (b_1, \ldots, b_n) \to (\ldots, a_s, b_t, \ldots)$$

goal: $\forall$correct algorithm $\mathcal{P}, a_s$ and b_t are compared in $\mathcal{P}$

$$\forall \mathcal{P}, (\mathcal{P} \text{ is correct }) \implies (a_s \text{ and } b_t \text{ are compared})$$

$$\forall \mathcal{P}, (a_s \text{ and } b_t \text{ are } \textbf{not} \text{ compared}) \implies (\mathcal{P} \text{ is incorrect})$$

$$\begin{cases} \mathcal{P} \text{ is incorrect on } A, B; \\ \end{cases}$$

- Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

Proof.

$$A = (a_1, \dots, a_n), B = (b_1, \dots, b_n) \to (\dots, a_s, b_t, \dots)$$

goal: $\forall$correct algorithm $\mathcal{P}, a_s$ and b_t are compared in $\mathcal{P}$

$$\forall \mathcal{P}, (\mathcal{P} \text{ is correct }) \implies (a_s \text{ and } b_t \text{ are compared})$$

$$\forall \mathcal{P}, (a_s \text{ and } b_t \text{ are } \textbf{not} \text{ compared}) \implies (\mathcal{P} \text{ is incorrect})$$

$$\begin{cases} \mathcal{P} \text{ is incorrect on } A, B; \\ \mathcal{P} \text{ is correct on } A, B, \text{then it's not correct on } A', B'. \end{cases}$$

$$A' = (a_1, \dots, a_{s-1}, b_t, a_{s+1}, \dots, a_n)$$

$$B' = (b_1, \dots, b_{t-1}, a_s, b_{t+1}, \dots, b_n)$$

$\square$

- Show a lower bound of $2n-1$ comparsions for merging two sorted lists.

- Show a lower bound of $2n-1$ comparsions for merging two sorted lists.

Proof.

$$A = (1, 3, \ldots, 2n-1), B = (2, 4, \ldots, 2n)$$

$\square$

- Show a lower bound of $2n-1$ comparsions for merging two sorted lists.

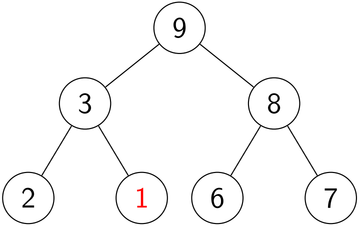
Proof.

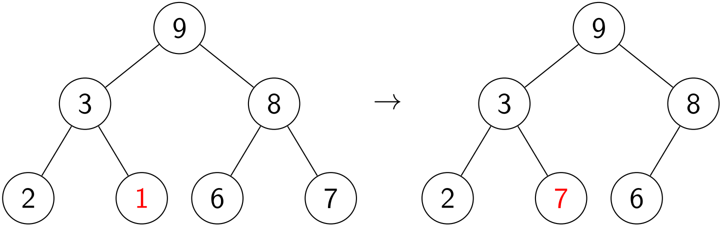
$$A = (1, 3, \ldots, 2n-1), B = (2, 4, \ldots, 2n)$$

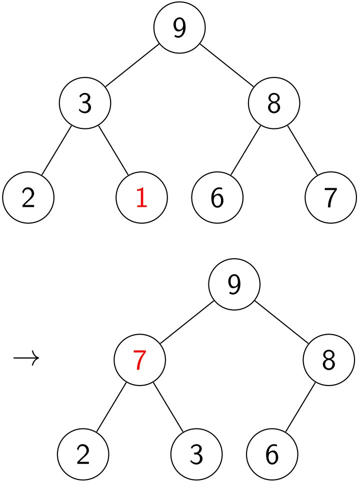
$\square$

Ex. (6.5.8 Heap Delete)

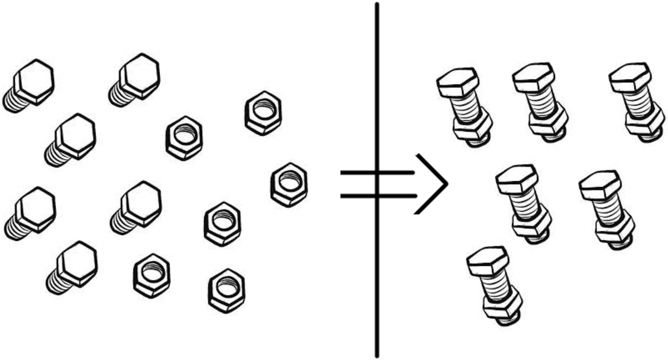
The operation HEAP-DELETE(A, i) deletes the item in node i from heap A. Give an implementation of HEAP-DELETE that runs in $O(\lg n)$ time for an n-element max-heap.




9
3
8
2
1
6
7
→
9
3
8
2
7
6


9
3 8
2 1 6 7
→
9
3 8
2 7 6
→
9
7 8
2 3 6

Ex. (Problem set 4, Additional Problem One, Nuts and Bolts)



- Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to correctly match up the nuts and bolts.

- Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to correctly match up the nuts and bolts.

Proof.

$\log n! = \Omega(n \log n)$ $\square$

- Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to correctly match up the nuts and bolts.

Proof.

$\log n! = \Omega(n \log n)$ □

- Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to find $n/2$ arbitrary matching nut-bolt pairs.

- Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to correctly match up the nuts and bolts.

Proof.
$\log n! = \Omega(n \log n)$ $\square$

- Prove that in the worst case, $\Omega(n \log n)$ nut-bolt tests are required to find $n/2$ arbitrary matching nut-bolt pairs.

Proof.
$\log \frac{n!}{(n/2)!} \geq \log(n/2)! = \Omega(n \log n)$ $\square$

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Proof.

$\log \frac{n!}{(n-k)!} = \cdots = \Omega(n + k \log n)$ □

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Proof.

$$\log \frac{n!}{(n-k)!} = \cdots = \Omega(n + k \log n) \qquad \square$$

When $k = 1 \rightarrow \log \frac{n!}{(n-1)!} = \log n = o(n + \log n)$

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Proof.

$\log \frac{n!}{(n-k)!} = \cdots = \Omega(n + k \log n)$ □

When $k = 1 \rightarrow \log \frac{n!}{(n-1)!} = \log n = o(n + \log n)$

- Prove that in the worst case, $\Omega(n + k\log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Proof.

$\log \frac{n!}{(n-k)!} = \cdots = \Omega(n + k\log n)$ ×

When $k = 1 \rightarrow \log \frac{n!}{(n-1)!} = \log n = o(n + \log n)$

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Ex. (3.1-1)

$\max(f(n), g(n)) = \Theta(f(n) + g(n))$

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Ex. (3.1-1)

$$\max(f(n), g(n)) = \Theta(f(n) + g(n))$$

$$\begin{cases} T(n) = \Omega(k \log n) \\ T(n) = \Omega(n) \end{cases} \implies T(n) = \Omega(n + k \log n)$$

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Ex. (3.1-1)

$\max(f(n), g(n)) = \Theta(f(n) + g(n))$

$$\begin{cases} T(n) = \Omega(k \log n) \\ T(n) = \Omega(n) \end{cases} \implies T(n) = \Omega(n + k \log n)$$

$$T(n) \geq \log \frac{n!}{(n-k)!} = \Omega(k \log n)$$

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Ex. (3.1-1)

$\max(f(n), g(n)) = \Theta(f(n) + g(n))$

$$\begin{cases} T(n) = \Omega(k \log n) \\ T(n) = \Omega(n) \end{cases} \implies T(n) = \Omega(n + k \log n)$$

$$T(n) \geq \log \frac{n!}{(n-k)!} = \Omega(k \log n) \checkmark$$

- Prove that in the worst case, $\Omega(n + k \log n)$ nut-bolt tests are required to find k arbitrary matching pairs.

Ex. (3.1-1)

$\max(f(n), g(n)) = \Theta(f(n) + g(n))$

$$\begin{cases} T(n) = \Omega(k \log n) \\ T(n) = \Omega(n) \end{cases} \implies T(n) = \Omega(n + k \log n)$$

$$T(n) \geq \log \frac{n!}{(n-k)!} = \Omega(k \log n) \checkmark$$

$$T(n) \stackrel{?}{=} \Omega(n)$$

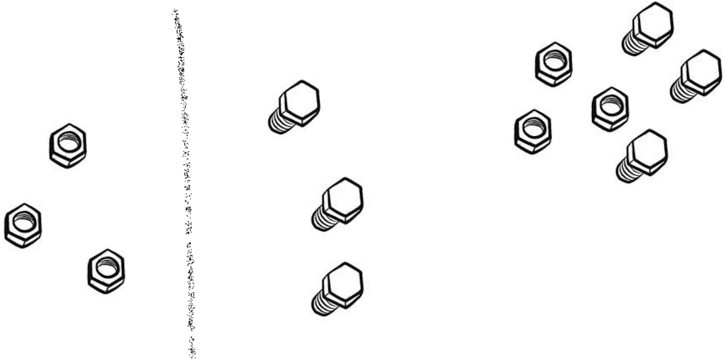
$T(n) \geq n/2$: Adversary argument

$T(n) \geq n/2$: Adversary argument

For any query, the adversary always answers “The nut is smaller then the bolt”.

$T(n) \geq n/2$: Adversary argument

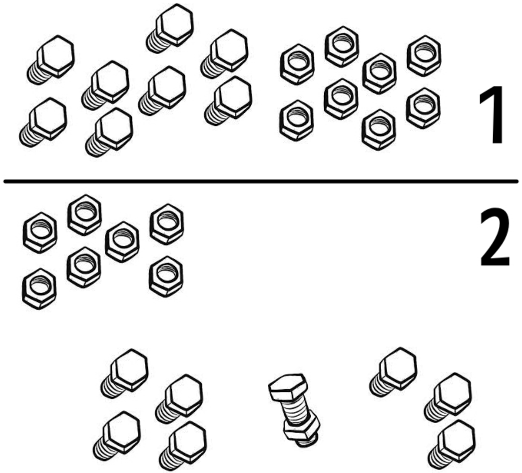
For any query, the adversary always answers "The nut is smaller then the bolt".



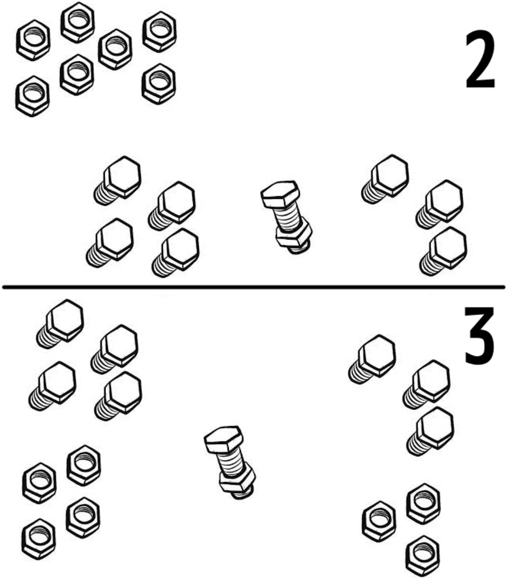
- Describe a randomized algorithm that finds k matching nut-bolt pairs in $O(n + k \log n)$ expected time.

- Find all nut-bolt pairs: Expected $O(n \log n)$

- Find all nut-bolt pairs: Expected $O(n \log n)$



- Find all nut-bolt pairs: Expected $O(n \log n)$



- Find all nut-bolt pairs: Expected $O(n \log n)$

$$\mathbb{E}[T(n)] = \sum_{i=1}^{n} \frac{1}{n} \Big(\mathbb{E}[T(i-1)] + \mathbb{E}[T(n-i)] + 2n \Big)$$

- Find all nut-bolt pairs: Expected $O(n \log n)$

$$\mathbb{E}[T(n)] = \sum_{i=1}^{n} \frac{1}{n} \Big(\mathbb{E}[T(i-1)] + \mathbb{E}[T(n-i)] + 2n \Big)$$

For quick sort $S(n)$:

$$\mathbb{E}[S(n)] = \sum_{i=1}^{n} \frac{1}{n} \Big(\mathbb{E}[S(i-1)] + \mathbb{E}[S(n-i)] + n \Big)$$

- Find all nut-bolt pairs: Expected $O(n \log n)$

$$\mathbb{E}[T(n)] = \sum_{i=1}^{n} \frac{1}{n} \Big(\mathbb{E}[T(i-1)] + \mathbb{E}[T(n-i)] + 2n \Big)$$

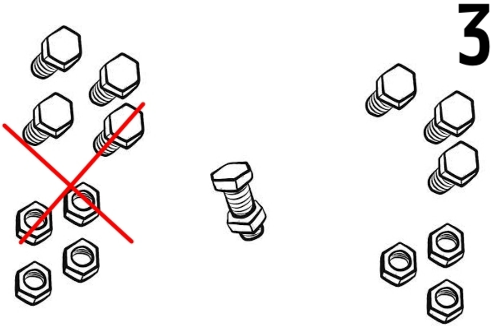
For quick sort $S(n)$:

$$\mathbb{E}[S(n)] = \sum_{i=1}^{n} \frac{1}{n} \Big(\mathbb{E}[S(i-1)] + \mathbb{E}[S(n-i)] + n \Big)$$

$$\mathbb{E}[T(n)] = \Theta(n \log n)$$

- Find k nut-bolt pairs: expected $O(n + k \log k)$

- Find k nut-bolt pairs: expected $O(n + k \log k)$



$$T(n) = \begin{cases} n \log n & \text{if } n < 2k \\ T(n/2) + 2n & \text{otherwise} \end{cases} \implies T(n) = \Theta(n + k \log k)$$

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Algorithm $\mathcal{A}$ on inputs $\mathcal{I}$ of size n

	O	Ω	Θ
Best-Case			
Wors-Case			

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Algorithm $\mathcal{A}$ on inputs $\mathcal{I}$ of size n

	O	Ω	Θ
Best-Case	by example		
Wors-Case			

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Algorithm $\mathcal{A}$ on inputs $\mathcal{I}$ of size n

	O	Ω	Θ
Best-Case	by example	"weakness" of A	
Wors-Case			

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Algorithm $\mathcal{A}$ on inputs $\mathcal{I}$ of size n

	O	Ω	Θ
Best-Case	by example	"weakness" of A	$O = \Omega$
Wors-Case			

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Algorithm $\mathcal{A}$ on inputs $\mathcal{I}$ of size n

	O	Ω	Θ
Best-Case	by example	"weakness" of A	$O = \Omega$
Wors-Case	"power" of A		

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Algorithm $\mathcal{A}$ on inputs $\mathcal{I}$ of size n

	O	Ω	Θ
Best-Case	by example	"weakness" of A	$O = \Omega$
Wors-Case	"power" of A	by example	

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Algorithm $\mathcal{A}$ on inputs $\mathcal{I}$ of size n

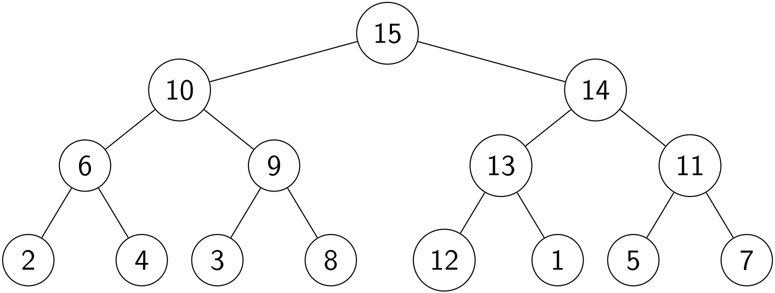
	O	Ω	Θ
Best-Case	by example	"weakness" of A	$O = \Omega$
Wors-Case	"power" of A	by example	$O = \Omega$

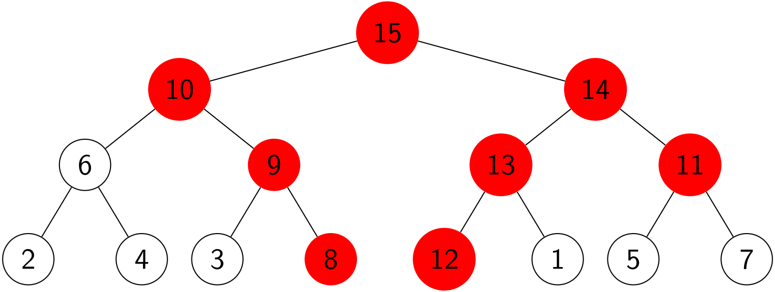
Ex. (6.4-5)

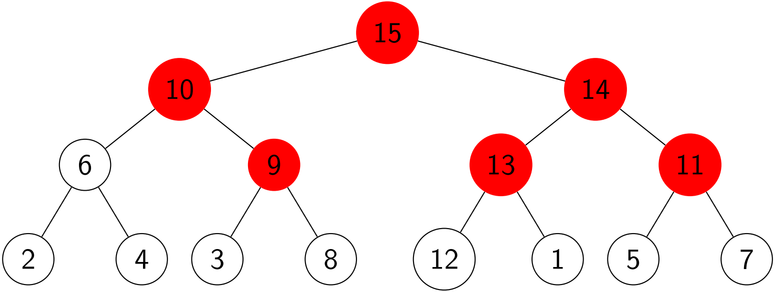
Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.


15
10
14
6
9
13
11
2
4
3
8
12
1
5
7


15
10
14
6
9
13
11
2
4
3
8
12
1
5
7


15
10
14
6
9
13
11
2
4
3
8
12
1
5
7

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

The largest $n/2$ elements form a subtree

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

The largest $n/2$ elements form a subtree

$\geq n/4$ must not be leaves

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

The largest $n/2$ elements form a subtree

$\geq n/4$ must not be leaves

$\geq n/8$ must have height $\geq \log n - 3$

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

The largest $n/2$ elements form a subtree

$\geq n/4$ must not be leaves

$\geq n/8$ must have height $\geq \log n - 3$

$$T(n) \geq (n/8)(\log n - 3)$$

Ex. (6.4-5)

Show that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

The largest $n/2$ elements form a subtree

$\geq n/4$ must not be leaves

$\geq n/8$ must have height $\geq \log n - 3$

$$T(n) \geq (n/8)(\log n - 3) = \Omega(n \log n)$$