Exercise sheet 3.

Advanced Algorithms

WiSe 2020/21

Instructor: László Kozma

Due 12:00, November 27th, 2020

Exercise 1 Selection *6 Points*

(a) (3 pts) Consider the median-of-medians selection algorithm seen in class, but with groups of seven instead of five. Analyze the running time of this algorithm.

(b) (3 pts) What is the exact number of comparisons necessary to find the median of five elements in the worst case? (Drawing a decision-tree may help.)

(c) **(Extra 6 pts)** Implement median-of-medians selection with groups of $3, 5, 7, 9, \ldots$ and run experiments. Compare experimentally the efficiency of the different variants on different input sizes, and report your findings.

Exercise 2 Dropping mobile phones *10 Points*

Suppose you live in a skyscraper with n floors and you want to test the durability of your mobile phone, i.e. the highest floor from which it can fall without breaking. You may assume that there is an exact threshold $0 \leq k \leq n$ such that dropping a phone from the $k+1$-th floor (or higher) breaks it but dropping it from the k-th floor (or lower) does not break it. You may also assume that if a phone falls without breaking, then its structure is not weakened (and it is not strengthened either). Once a phone is broken, it cannot be used for further tests.

(a) (2 pts) If you have only one phone to experiment with (that you are willing to break), what is the worst-case number of phone-drops necessary to find the threshold value k?

(b) (3 pts) Suppose now that you have two (identical) phones. Show that in this case $O(\sqrt{n})$ phone-drops are sufficient.

(c) (3 pts) Suppose that you have t identical phones. Show that in this case, $O(tn^{1/t})$ phone-drops are sufficient. What happens when $t \geq \lceil \log_2 n \rceil$?

(d) (2 pts) Can you obtain the same bounds, but with "n" replaced by "k"? (This would be an improvement, as k may be much smaller than n.)

Exercise 3 Second largest element *7 Points*

(a) (4 pts) Give an algorithm, that finds both the largest and the second largest among n distinct elements, with $n + \lceil \log_2 n \rceil - 2$ comparisons. (The trivial method would require $2n - 3$ comparisons in the worst case.)

(b) (3 pts) Show a non-trivial lower bound on the number of comparisons required in the worst case. (For extra credit, show that the above number of comparisons is optimal for any deterministic algorithm.)

(c) **(Extra 5 pts)** Find a *randomized* algorithm that solves this task with fewer comparisons in expectation—this may be tricky.

Exercise 4 Popular item *4 Points*

Given a list of n items (with possible repetitions), a *popular item* is one that occurs at least $\lceil n/2 \rceil$ times in the list. Suppose we are allowed to use only *equality-queries*, i.e. finding out whether two elements are equal or not. Describe an algorithm for finding a popular element (or reporting that none exists) with $O(n)$ such queries.

(Optional:) can you extend your algorithm to find an element occuring at least $\lceil \alpha \cdot n \rceil$ times, for given fixed $0 < \alpha < 1$?

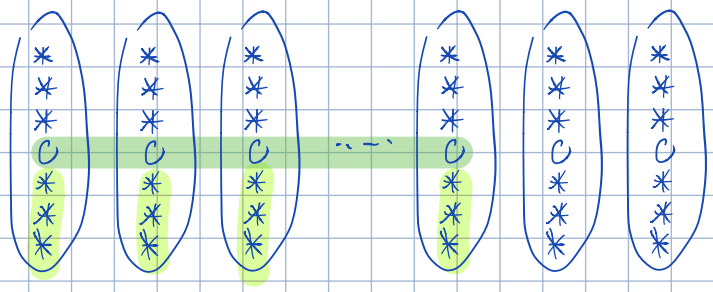
Total: 27 points. Have fun with the solutions!

From: Yumeng Li and Thore Brehmer

Exercise 1 Selection *6 Points*

(a) (3 pts) Consider the median-of-medians selection algorithm seen in class, but with groups of seven instead of five. Analyze the running time of this algorithm.

1 a)



$\frac{n}{7}$ many groups

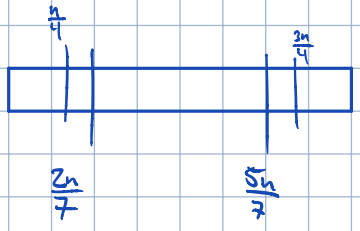
smaller-medians $= \frac{n}{7} \cdot \frac{1}{2} = \frac{n}{14}$

smaller-elements-of-smaller-medians $= \left(\frac{n}{7} \cdot \frac{1}{2}\right) \cdot 3 = \frac{3n}{14}$

$\#$ => atleast $\frac{4n}{14}$ smaller elements

$\#$ => atleast $\frac{4n}{14}$ bigger elements

A



1a)

Running time

$G(n)$

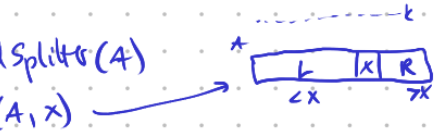
GoodSplitter(A)

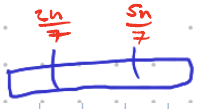
$c \cdot n$	1. Split A into groups of ~~five~~ seven
$C \cdot n$	2. Find media of each group (write them in a new group M) $M = (m_1, \dots, m_{\lceil \frac{n}{7} \rceil})$
$T(\frac{n}{7})$	3. $m \leftarrow$ median (M) $(\text{Select}(M, \lfloor \frac{\lvert M \rvert + 1}{2} \rfloor))$
$O(1)$	4. return (m)

$T(n)$

Select(A, k)

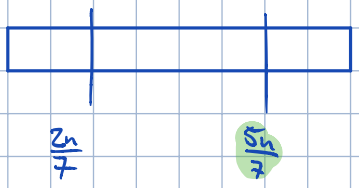
$T'(n)$	$x \leftarrow$ GoodSplitter(A)
$c \cdot n$	Partition (A, x)
$O(1)$	if $\lvert L \rvert = k-1$ then return x
$T(\lvert L \rvert)$	if $\lvert L \rvert \geq k$ then return Select (L, k)
$T(\lvert R \rvert)$	(if $\lvert L \rvert < k-1$) return Select $(R, k - \lvert L \rvert - 1)$





$$G(n) \leq c^* \cdot n + T(\tfrac{n}{7})$$

$$T(n) \leq G(n) + c' \cdot n + T(\tfrac{5n}{7})$$



$$T(n) \leq c' \cdot n + c^* \cdot n + T(\tfrac{n}{7}) + T(\tfrac{5n}{7})$$

Guess: $T(n') \leq 7 c'' n'$ for all $n' < n$

$$T(n) \overset{(IH)}{\leq} c'' n + 7c'' \cdot \tfrac{n}{7} + 7c'' \cdot \tfrac{5n}{7}$$

$$= 7 c'' n \quad \checkmark$$

$$\Rightarrow T(n) \in O(n)$$

1b) (b) (3 pts) What is the exact number of comparisons necessary to find the median of five elements in the worst case? (Drawing a decision-tree may help.)

Ex1b $A = (a, b, c, d, e)$

① compare a & b, c & d, assume $a<b$, $c<d$ #comparison=2

② compare b & d 1 comparison

WLOG assume $b<d \Rightarrow \begin{cases} a<b<d \\ c<d. \end{cases}$ so d can't be median

③ insert e to $a<b<d$, compare e with b.

if $e<b$, then compare e and a

if $e>b$, then compare e and d. 2 comparisons.

④ if $a<e<b<d$,

compare b with c

since $c<d$,

the larger of "b and c"

is median.

In total 6 comparisons.

| if $a<b<e<d$.

| Compare c with e 1 comparison.

| since $c<d$

| the larger of "c and e" is median.

(c) **(Extra 6 pts)** Implement median-of-medians selection with groups of $3, 5, 7, 9, \ldots$ and run experiments. Compare experimentally the efficiency of the different variants on different input sizes, and report your findings.

1c)

Comparisons

$T(n)$

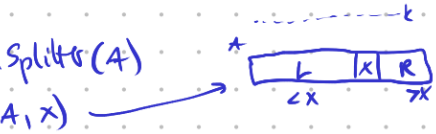
GoodSplitter (A)

O	1. Split A into groups of k
$\lceil \frac{n}{k} \rceil \cdot k \log k$	2. Find media of each group $M = (m_1, \ldots, m_{\lceil \frac{n}{5} \rceil})$
$T(\frac{n}{k})$	3. $m \leftarrow$ median (M) $(\text{Select}(M, \lfloor \frac{\lvert M \rvert + 1}{2} \rfloor))$
O	4. return (m)

$T(n)$

Select (A, k)

$T(n)$	
n	$x \leftarrow$ GoodSplitter (A)
	Partition (A, x)
1	if $\lvert L \rvert = k-1$ then return x
1	if $\lvert L \rvert \geq k$ then return Select (L, k)
1	(if $\lvert L \rvert < k-1$) return Select $(R, k - \lvert L \rvert - 1)$



- In our implementation the "Find median of each group" will take $\lceil \frac{n}{k} \rceil \cdot k \log_2 k$ comparisons.
  - $\lceil \frac{n}{k} \rceil$: because there will be $\lceil \frac{n}{k} \rceil$ groups
  - $k \log_2 k$: because we use quicksort to find the median of the group with k items

```
thob97@andorra:~/uni/hohere_alg$ python3 2.py
Random array length= 1000 ; number of tests= 100
group_size: 3 , Average_Comparisons: 8800.84 , Exec_Time: 0.8534862995147705
group_size: 5 , Average_Comparisons: 9141.81 , Exec_Time: 0.6771280765533447
group_size: 7 , Average_Comparisons: 7820.24 , Exec_Time: 0.5248019695281982
group_size: 9 , Average_Comparisons: 8685.75 , Exec_Time: 0.5310769081115723
group_size: 11 , Average_Comparisons: 8526.61 , Exec_Time: 0.5016462802886963
thob97@andorra:~/uni/hohere_alg$ python3 2.py
Random array length= 1000 ; number of tests= 100
group_size: 3 , Average_Comparisons: 8839.01 , Exec_Time: 0.8518011569976807
group_size: 5 , Average_Comparisons: 9163.04 , Exec_Time: 0.6772277355194092
group_size: 7 , Average_Comparisons: 7728.72 , Exec_Time: 0.5367679595947266
group_size: 9 , Average_Comparisons: 8614.71 , Exec_Time: 0.5273675918579102
group_size: 11 , Average_Comparisons: 8530.76 , Exec_Time: 0.4987208843231201
thob97@andorra:~/uni/hohere_alg$ python3 2.py
Random array length= 1000 ; number of tests= 100
group_size: 3 , Average_Comparisons: 8919.84 , Exec_Time: 0.8476605415344238
group_size: 5 , Average_Comparisons: 9168.91 , Exec_Time: 0.7364740371704102
group_size: 7 , Average_Comparisons: 7776.4 , Exec_Time: 0.5517847537994385
group_size: 9 , Average_Comparisons: 8656.19 , Exec_Time: 0.5148868560791016
group_size: 11 , Average_Comparisons: 8542.54 , Exec_Time: 0.4895081520080566
thob97@andorra:~/uni/hohere_alg$
```

So it seems like that with group_size = 7 it make the least number of comparisons. But larger group-sizes will be faster in executen_time

NO.
DATE

§3

(2) a) The worst-case number of phone drops is K.
Since I have only one phone, I have to test one by one.

b) Since I have two phones. I can use the first phone to test the scope and then use the second phone to test one by one within the scope.
say: check t-th ~~flour~~ floor, $2t$-th, $\cdots$, $\lfloor \frac{n}{t} \rfloor t$, n in this order. find i s.t. it-th floor the first phone breaks.
Then use the second phone to test $[(i-1)t, it]$ floors.
Time: $\frac{n}{t} + t$
optimum: $t = \sqrt{n}$, then total time is $2\sqrt{n} \in O(\sqrt{n})$

c) Use the first phone to test every x_1 floors, i.e. $x_1, 2x_1, \cdots$
Suppose the first phone breaks at ~~$i x_1$ floors~~ (i, x_1) floor.
Then use the second phone to test every x_2 floors in the scope $[(i-1)x_1, i x_1]$, i.e. $(i-1)x_1 + x_2$, $(i-1)x_1 + 2x_2$, $\cdots$
$\vdots$ the same to the t phones. t-phone should test one by one.

$$T(n) = \frac{n}{x_1} + \frac{x_1}{x_2} + \frac{x_2}{x_3} + \cdots + \frac{x_{t-2}}{x_{t-1}} + x_{t-1}$$

$$f(x_1, \cdots, x_t) = \frac{n}{x_1} + \frac{x_1}{x_2} + \cdots + \frac{x_{t-2}}{x_{t-1}} + x_{t-1}$$

$$0 = f'_{x_1}(x_1, \cdots, x_t) = -\frac{n}{x_1^2} + \frac{1}{x_2}$$

$$0 = f'_{x_2}(x_1, \cdots, x_t) = -\frac{x_1}{x_2^2} + \frac{1}{x_3} \qquad 0 = f'_{x_{t-2}}(x_1, \cdots, x_t) = -\frac{x_{t-3}}{x_{t-2}^2} + \frac{1}{x_{t-1}}$$

$$0 = f'_{x_{t-1}}(x_1, \cdots, x_t) = -\frac{x_{t-2}}{x_{t-1}^2} + 1.$$

~~$f'_{x_t}(x_1, \cdots, x_t) = 1 \Rightarrow x_{t-1} = \sqrt{x_{t-2}}$~~

~~$x_{t-2} = (x_{t-3})^{\frac{1}{2}}$~~

~~$x_{t-2} = \left(\frac{x_{t-3}}{x_{t-1}}\right)^{\frac{1}{2}} = (x_{t-3})^{\frac{1}{2}} (x_{t-1})^{-\frac{1}{2}} = (x_{t-3})^{\frac{1}{2}} (x_{t-2})^{-\frac{1}{4}}$~~

$\Rightarrow x_{t-1} = x_{t-2}^{\frac{1}{2}}$
$x_{t-2} = x_{t-3}^{\frac{2}{3}}$
$x_{t-3} = x_{t-4}^{\frac{3}{4}}$
$\vdots$
$x_2 = x_1^{\left(\frac{t-2}{t-1}\right)}$
$x_1 = n^{\frac{t-1}{t}}$

$x_2 = n^{\frac{t-1}{t}}$, $x_3 = n^{\frac{t-3}{t}}$, $\cdots$, $x_{t-1} = n^{\frac{1}{t}}$

optimum $\Rightarrow$ $T(n) = \frac{n}{x_1} + \frac{x_1}{x_2} + \cdots + \frac{x_{t-2}}{x_{t-1}} + x_{t-1} = n^{\frac{1}{t}} + n^{\frac{1}{t}} + \cdots + n^{\frac{1}{t}} = t\, n^{\frac{1}{t}}$.

NO.
DATE

c) When $t \geq \lceil \log_2 n \rceil$, we have enough phones to use binary search to test.

d). As analyzed in (b) and (c), ~~we explain by using two phones~~.

- We check 0-th, t_1-th, $2t_1$-th, ..., floors in order. Suppose k is between $[it_1, (i+1)t_1]$, then we only test $(i+1)$ times. $(i+1) = \lceil \frac{k}{t_1} \rceil$.
- Then we use the second phone to check in order:
  $it_1 + t_2$, $it_1 + 2t_2$, ..., $it_1 + \lceil \frac{k-it_1}{t_2} \rceil t_2$.
- use the third phone to check
  $it_1 + \lfloor \frac{k-it_1}{t_2} \rfloor t_2 + t_3$, $it_1 + \lfloor \frac{k-it_1}{t_2} \rfloor t_2 + 2t_3$, ...
  $it_1 + \lfloor \frac{k-it_1}{t_2} \rfloor t_2 + \lceil \frac{k-it_1 - \lfloor \frac{k-it_1}{t_2} \rfloor t_2}{t_3} \rceil t_3$

$\Rightarrow f(1) = \lceil \frac{k}{t_1} \rceil$, $f(2) = \frac{\lceil k - (f(1)-1)t_1 \rceil}{t_2} \leq \frac{t_1}{t_2}$.

$f(3) = \lceil \frac{k - (f(1)-1)t_1 - (f(2)-1)t_2}{t_3} \rceil \leq \frac{t_2}{t_3}$.

Obs: these are the same to those in (c) by replacing n by k. using the same method to solve it.

$T(n) = f(1) + f(2) + f(3) + \cdots$

$= tk^{\frac{1}{t}}$.

Exercise 3 Second largest element *7 Points*

(a) (4 pts) Give an algorithm, that finds both the largest and the second largest among n distinct elements, with $n + \lceil \log_2 n \rceil - 2$ comparisons. (The trivial method would require $2n - 3$ comparisons in the worst case.)

(b) (3 pts) Show a non-trivial lower bound on the number of comparisons required in the worst case. (For extra credit, show that the above number of comparisons is optimal for any deterministic algorithm.)

(c) **(Extra 5 pts)** Find a *randomized* algorithm that solves this task with fewer comparisons in expectation—this may be tricky.

a)

```
def second_largest(A'):
    A = [[i] for i in range(0, len(A'))]
    for i in range(0, len(A), 2):
        if A[i][0] > A[i+1][0]:
            A = A + [A[i] + A[i+1][0]]
        else:
            A = A + [A[i+1] + A[i][0]]

    max = A[-1][0]
    A = A[-1][1:]
    for i in range(0, len(A), 2):
        if A[i] > A[i+1]:
            A = A + A[i]
        else:
            A = A + A[i+1]

    second_max = A[-1]
    return (max, second_max)
```

$(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots) - 1 = n - 1$

$|A| = \lceil \log_2 n \rceil - 1$

e.g.

$A' = [2, 5, 3, 7]$

$\Rightarrow A = [[2], [5], [3], [7]]$

$A = [[2], [5], [3], [7]] + [5, 2] + [3, 7] + [3, 7, 5]$

$A = [[2], [5], [3], [7], [5, 2], [3, 7], [3, 7, 5]]$ (max)

$A = [7, 5]$

$A = [7, 5] + [7]$

$A = [7, 5, 7]$ (second-max)

first Loop $n-1$ comparisons

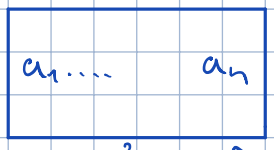
second Loop $\log_2 n - 1$ comparisons

$\Rightarrow$ $n + \log_2 n - 2$ comparisons total

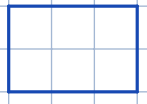
b)

Adversary groups elements into

Not-Yet-Decided



$a_i \overset{?}{\geq} a_j$



Losers

Initially $a_1, \dots a_n$ are
- "Not-Yet-Decided"
- and every element has a counter

Possible queries

NYD vs NYD:

if. x.counter = y.counter and $x < y$

Losers — y.counter += 1

L vs L
L vs NYD
} arbitrary

Algorithm can terminate if $|\text{"Not-Yet-Decided"}| \leq 2$

if $|N\text{-}Y\text{-}D| = 1$ $\Rightarrow$ last element is max

if $|N\text{-}Y\text{-}D| = 2$ $\Rightarrow$ adversary has to compare them $\Rightarrow$ pick max
(for case n = odd then the last elements might have different counters)

$\Rightarrow$ the adversary can force $\frac{n}{2} + \frac{n}{4} + \dots - 1 = n-1$ comparisons for max

- to find the second largest element we just have to compare the elements which the max won against.
- These will be $\log_2 n$ elements. $\Rightarrow \log n - 1$ comparisons needed to find the (second) biggest element

↳ • this is because we know from lecture that it takes $n-1$ comparisons to find the max.

• And because the adversery compared elements in such a way that it can be seen as balanced tree. The height of that tree is $\log n$

⇒ The max got compared with $\log_2 n$ other elements.

⇒ second largest element = $\log_2 n - 1$ comparisons

in total $\log_2 n - 1 + n - 1 = n + \lceil \log_2 n \rceil - 2$

Exercise 4 Popular item *4 Points*

Given a list of n items (with possible repetitions), a *popular item* is one that occurs at least $\lceil n/2 \rceil$ times in the list. Suppose we are allowed to use only *equality-queries*, i.e. finding out whether two elements are equal or not. Describe an algorithm for finding a popular element (or reporting that none exists) with $O(n)$ such queries.

(Optional:) can you extend your algorithm to find an element occuring at least $\lceil \alpha \cdot n \rceil$ times, for given fixed $0 < \alpha < 1$?

```
def find_popular (liste):
    n = len(liste)                      O(1)
    dict = {}                           O(1)
    for i in liste:                     O(n)
        dict[i] = 0                     O(1)
    for i in liste:                     O(n)
        dict[i] = dict[i] + 1           O(1)
        if dict[i] == round(n/2.0):     O(1)
            return i                    O(1)
    return None                         O(1)
```

$\Rightarrow Z_n \in O(n)$

e.g. liste = [0, 0, 0, 1, 1, 1, 1]

dict = {0:0, 1:0}

dict = {0:3, 1:4}