Exercise sheet 2.

Advanced Algorithms

WiSe 2020/21

Instructor: László Kozma

Due 12:00, November 20th, 2020

Exercise 1 Recurrences

6 × 2 = 12 Points

Find sharp asymptotic upper bounds (using your favorite method) for $T(n)$ in each of the following cases. You may assume that $T(n) < 10$ for all $0 \le n \le 5$.

(a) $T(n) = 3 \cdot T(\lfloor n/3 \rfloor) + n$,

(b) $T(n) = 4 \cdot T(\lfloor n/2 \rfloor) + n^2$,

(c) $T(n) = T(\lfloor n/2 \rfloor) + n \log_2 n$,

(d) $T(n) = n \cdot T(\lfloor n/2 \rfloor) + 1$,

(e) $T(n) = 2^n \cdot T(\lfloor 2n/3 \rfloor)$,

(f) $T(n) = T(n-1) + 1/n$,

(g) **(Bonus 2 pts)** $T(n) = T(n - \lfloor \sqrt{n} \rfloor) + 2n$.

Exercise 2 Min and Max

6 Points

(a) (3 pts) Describe an algorithm for finding both the minimum and the maximum of a list of n items from an ordered set, using $\lceil 3n/2 \rceil + a$ comparisons. What is the best constant a you can achieve?

(b) (3 pts) Show a lower bound of $\lceil 3n/2 \rceil + b$ on the number of comparisons required in the worst-case (by every comparison-based algorithm). What is the best constant b you can achieve?

Exercise 3 Merging

3 Points

Suppose we want to merge two sorted lists, A (of length a) and B (of length b), where $a > b$. Recall that in the exercise session we argued that this can be achieved with $O(b \log \frac{a}{b})$ comparisons. (We used doubling binary search—recall this argument).

(a) Show that every comparison-based merging algorithm must do $\Omega(b \log \frac{a}{b})$ comparisons.
Hint: how many different "merges" can happen? Use the information-theory lower bound. You may use the inequality $\binom{n}{k} \ge (\frac{n}{k})^k$.

(b) **(Bonus 3 pts)** The *doubling binary search* allows us to find x in a sorted array, with at most $2\lceil \log_2 k \rceil + 1$ comparisons, where k is the position of x in the list. Can you improve this to $\log_2 k + o(\log(k))$ comparisons?

Exercise 4 Random shuffle *6 Points*

Below are three different algorithms that are each supposed to produce a permutation of an array $a[1], \ldots, a[n]$ *uniformly at random* (we want each permutation to be equally likely). Which of the three algorithms are correct? Motivate your answer. We assume that **RAND(i,j)** produces an integer uniformly at random from $i, \ldots, j$, and that **SWAP(i,j)** exchanges $a[i]$ and $a[j]$.

```
SHUFFLE1:
for i from 1 to n do
  j <- RAND(i,n)
  SWAP(i,j)

SHUFFLE2:
for i from 1 to n do
  j <- RAND(1,n)
  SWAP(i,j)

SHUFFLE3:
for i from 1 to n do
  j <- RAND(1,i)
  SWAP(i,j)
```

Exercise 5 Practice (optional) *0 Points*

Sort the array $[9, 1, 6, 7, 0, 8, 2, 4, 5, 3]$ by hand, using Mergesort (with simple, sequential merging) and deterministic Quicksort. How many comparisons are used by each?

Total: 27 points. Have fun with the solutions!

From: Yumeng Li and Thore Brehmer

Exercise 1 Recurrences *6 × 2 = 12 Points*

Find sharp asymptotic upper bounds (using your favorite method) for $T(n)$ in each of the following cases. You may assume that $T(n) < 10$ for all $0 \leq n \leq 5$.

(a) ~~$T(n) = 3 \cdot T(\lfloor n/3 \rfloor) + n$,~~

(b) ~~$T(n) = 4 \cdot T(\lfloor n/2 \rfloor) + n^2$,~~

(c) ~~$T(n) = T(\lfloor n/2 \rfloor) + n \log_2 n$,~~ testen

(d) $T(n) = n \cdot T(\lfloor n/2 \rfloor) + 1$, $n^n \log n$

(e) $T(n) = 2^n \cdot T(\lfloor 2n/3 \rfloor)$,

(f) ~~$T(n) = T(n-1) + 1/n$,~~ testen

(g) **(Bonus 2 pts)** $T(n) = T(n - \lfloor \sqrt{n} \rfloor) + 2n$.

Master Theorem

$\begin{cases} T(n) = a \cdot T(n/b) + c \cdot n^k \\ T(1) = c \end{cases}$

root dominates (1) $T(n) \in \Theta(n^k)$, if $a < b^k$

cost spread out across tree (2) $T(n) \in \Theta(n^k \log n)$, if $a = b^k$

base cases (leaves) dominate (3) $T(n) \in \Theta(n^{\log_b a})$, if $a > b^k$

a) $T(n) = 3 \cdot T(\lfloor n/3 \rfloor) + n$

$a = 3, b = 3, k = 1 \quad \dots \quad 3 = 3^1 \Rightarrow a = b^k$ (Master method)

$\Rightarrow n \log_3 n$

Testing: $\leq 3 \cdot \frac{n}{3} \log_3 \frac{n}{3} + n$

$= n(\log_3 n - \log_3 3) + n$

$= n \log_3 n$ ✓ 2p

b) $T(n) = 4 \cdot T(\lfloor n/2 \rfloor) + n^2$

$a = 4, b = 2, k = 2 \quad \dots \quad 4 = 2^2 \Rightarrow a = b^k$ (Master method)

$\Rightarrow n^2 \log_3 n$

Testing: $\geq 4 \cdot \left(\frac{n}{2}\right)^2 \log \frac{n}{2} + n^2$

$= n^2(\log n - \log_2 2) + n^2$

$= n^2 \log_2 n^2$ ✓ 2p

a bit nonstandard but o.k.

c) $T(n) = T(\lfloor n/2 \rfloor) + n \log_2 n$

$a = 1, b = 2, k = 1 \quad \dots \quad 1 < 2^1 \Rightarrow a < b^k$ (Master method)

$\Rightarrow n \log_2 n$ ✓ 2p

NO. 2nd HW

DATE

Ex 1) $T(n) < 10$, $0 \le n \le 5$

f) $T(n) = T(n-1) + \frac{1}{n}$

$T(n) = T(n-2) + \frac{1}{n-1} + \frac{1}{n} = T(5) + \frac{1}{6} + \cdots + \frac{1}{n} \le \int_5^n \frac{1}{x}\,dx + 10$

$= \ln(n) + 5$

$T(n) \ge \int_6^{n+1} \frac{1}{x}\,dx = \ln(n+1) - \ln 6$

$\Rightarrow T(n) \in O(\lg n)$ ← 2p.

e) $T(n) = 2^n \cdot T(\lfloor 2n/3 \rfloor)$

$T(n) = 2^n \cdot 2^{\lfloor \frac{2n}{3} \rfloor} T(\lfloor (\frac{2}{3})^2 n \rfloor) = \cdots \le 2^{n + \lfloor \frac{2n}{3} \rfloor + \lfloor (\frac{2}{3})^2 n \rfloor + \cdots + \lfloor (\frac{2}{3})^{\lfloor \log_{\frac{2}{3}} \frac{5}{n} \rfloor + 1} n \rfloor} T(5)$

where $(\frac{2}{3})^k n \le 5 \Rightarrow k \le \log_{\frac{2}{3}} \frac{5}{n}$

$n + \lfloor \frac{2n}{3} \rfloor + \lfloor (\frac{2}{3})^2 n \rfloor + \cdots + \lfloor (\frac{2}{3})^{\lfloor \log_{\frac{2}{3}} \frac{5}{n} \rfloor} n \rfloor$

$\le n + \frac{2n}{3} + \cdots + (\frac{2}{3})^{\log_{\frac{2}{3}} \frac{5}{n}} \cdot n = n \cdot \frac{(1 - (\frac{2}{3})^{\log_{\frac{2}{3}} \frac{5}{n} + 1})}{1 - \frac{2}{3}} = 3n - 10$

$\Rightarrow T(n) \le 10 \cdot 2^{3n-10}$

Given $T(n) \in O(2^{3n})$ suppose for all $k \le n-1$, $T(k) \in O(2^{3k})$

$T(n) = 2^n \cdot T(\lfloor \frac{2n}{3} \rfloor) \le 2^n \cdot 2^{\lfloor \frac{2n}{3} \rfloor \cdot 3} \le 2^n \cdot 2^{2n} \cdot c$ for some constant c so $T(n) \in O(2^{3n})$. ← 2p

d) $T(n) = n \cdot T(\lfloor \frac{n}{2} \rfloor) + 1$

$= n \cdot (\lfloor \frac{n}{2} \rfloor T(\lfloor (\frac{1}{2})^2 n \rfloor) + 1) + 1$

$= n \cdot \lfloor \frac{n}{2} \rfloor T(\lfloor (\frac{1}{2})^2 n \rfloor) + n + 1$

1.5p

$= n \cdot \lfloor \frac{n}{2} \rfloor (\lfloor (\frac{1}{2})^2 n \rfloor T(\lfloor (\frac{1}{2})^3 n \rfloor) + 1) + n + 1$

$= n \cdot \lfloor \frac{n}{2} \rfloor \lfloor (\frac{1}{2})^2 n \rfloor T(\lfloor (\frac{1}{2})^3 n \rfloor) + n \lfloor \frac{n}{2} \rfloor + n + 1$

$\cdots \le n \cdot \lfloor \frac{n}{2} \rfloor \lfloor (\frac{1}{2})^2 n \rfloor \cdots \lfloor (\frac{1}{2})^k n \rfloor \cdot T(5) + 1 + n + n \lfloor \frac{n}{2} \rfloor + \cdots + n \lfloor \frac{n}{2} \rfloor \lfloor \frac{n}{2^2} \rfloor \cdots \lfloor \frac{n}{2^{k-1}} \rfloor$

$(\frac{1}{2})^k n \le 5 \Rightarrow k \ge -\lg \frac{5}{n} = \lg \frac{n}{5}$

$T(n) \le n^k (\frac{1}{2})^{(1+2+\cdots+k-1)} T(5) + 1 + n + n^2 \cdot \frac{1}{2} + \cdots + n^{k-1} (\frac{1}{2})^{(1+2+\cdots+k-2)}$

$\le n^k (\frac{1}{2})^{\frac{k(k-1)}{2}} + n^{k-1} (\frac{1}{2})^{\frac{(k-1)(k-2)}{2}} + n^{k-2} (\frac{1}{2})^{\frac{(k-2)(k-3)}{2}} + \cdots + n + 1$

good, just plug in $k = \log_2 n$ to simplify

Exercise 2 Min and Max *6 Points*

(a) (3 pts) Describe an algorithm for finding both the minimum and the maximum of a list of n items from an ordered set, using $\lceil 3n/2 \rceil + a$ comparisons. What is the best constant a you can achieve?

(b) (3 pts) Show a lower bound of $\lceil 3n/2 \rceil + b$ on the number of comparisons required in the worst-case (by every comparison-based algorithm). What is the best constant b you can achieve?

a) $\lceil \frac{3n}{2} \rceil + a$ need $n-1$ comparisons to go trough the list

$\Rightarrow \lceil \frac{3n}{2} \rceil + a - \frac{2n}{2} - 1 = \lceil \frac{n}{2} \rceil + a - 1$ left

↳ "divide" the list

```
def min_max(list):
    for i in range(0, len(list), 2):          # 2 steps
        if list[i+1] < list[i]:               # max: list[i+1], min: list[i]
            list[i], list[i+1] = list[i+1], list[i]

    min = list[0]
    max = list[1]
    for i in range(2, len(list), 2):

        if ( list[i] < min ):
            min = list[i]

        if ( list[in] < max ):
            max = list[in]
```

list loop: $\frac{n}{2} \cdot 1$ comparisons

list loop: $\frac{n}{2} \cdot 2$ comparisons; loop starts at 2: -2

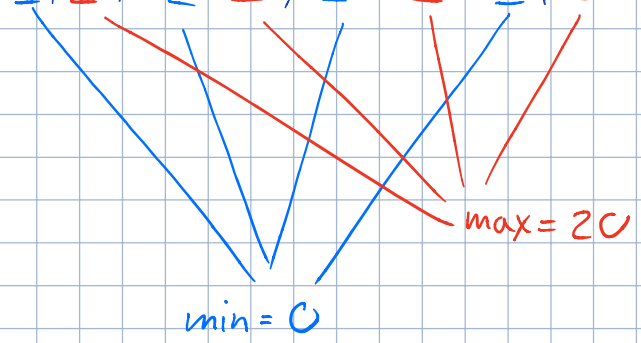
$\Rightarrow \frac{n}{2} + n - 2 = \lceil \frac{3n}{2} \rceil - 2 \Rightarrow \underline{a = -2}$

3p

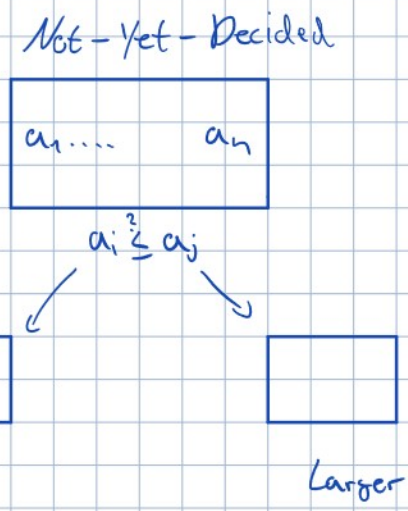
e.g. List = [5, 3, 10, 9, 2, 20, 0, 8]

first loop = [3, 5, 9, 10, 2, 20, 0, 8]

second loop = [3, 5, 9, 10, 2, 20, 0, 8]



b) Adversary groups elements into



Initialy $a_1, \dots a_n$ are "Not-Yet-Decided"

Possible queries

NYD vs NYD: arbitrary

$x \leq y$

↙ smaller ↘ larger

Smaller: (was smaller than some other element ⇒ can not be max ⇒ but can be min)

Larger: (was bigger than some other element ⇒ can not be min ⇒ but can be max)

S vs S
L vs L
L vs S
L vs NYD
S vs NYD
} arbitrary

Algorithm can terminate if "Not-Yet-Decided" is empty and $|Smaller| = 1$ and $|Larger| = 1$

only if $|N\text{-}y\text{-}D| = 0$
initialy $|N\text{-}y\text{-}D| = n$
} adversary can force $\frac{n}{2}$ comparisons

(if $|N\text{-}y\text{-}D| = 1$, then put the last element in smaller or Lager. It doesen't matter)

smaller and bigger will both be size $\frac{n}{2}$ now.

From lecture we know it takes $n-1$ comparisons to find the minimum of a list. ⇒ it also takes $n-1$ comparisons to find the maximum of a list. → true, but we should be careful not to assume algorithm proceeds via these steps (argument should work for any algorithm)

build smaller and bigger with $\frac{n}{2}$ comparisons

$|smaller| = \frac{n}{2}$ find min ⇒ $\frac{n}{2} - 1$ comparisons

$|Lager| = \frac{n}{2}$ find max ⇒ $\frac{n}{2} - 1$ comparisons

so in total $2(\frac{n}{2} - 1) + \frac{n}{2} = \lceil \frac{3n}{2} \rceil - 2$ comparisons ⇒ $a = -2$ ← 3P

3) a) There are in total $\binom{a+b}{b}$ merges. And consider the decision tree, which has $\binom{a+b}{b}$ leaves. So the depth is at least $\log_2 \binom{a+b}{b}$. depth = # comparisons

$\log_2 \binom{a+b}{b} \geq \log_2 \left(\frac{a+b}{b}\right)^b = b \log_2 \left(\frac{a}{b}+1\right) \in \Omega(b \log_2 \frac{a}{b})$ 3p

b) binary search : check elements with index

$1, 2, 2^2, 2^3, \ldots, 2^{\lceil \log k \rceil}$

- Improve by searching elements with index

$2^1, 2^{2^1}, 2^{2^2}, 2^{2^3}, \ldots, 2^{2^{\lceil \log \log k \rceil}}$ $\rightarrow O(\log \log k)$

and find $[2^{2^t}, 2^{2^{t+1}}] \ni 2^{\lceil \log k \rceil}$

- Then we do binary search of elements of index 2^i in $[2^{2^t}, 2^{2^{t+1}}]$

$2^t \leq i \leq 2^{t+1}$, $t \leq \lceil \log \log k \rceil$

These are at most 2^t elements.

cost of this binary search $= t \leq \lceil \log \log k \rceil$.

- binary search in $[2^{\lceil \log k \rceil - 1}, 2^{\lceil \log k \rceil}]$ $\rightarrow \log k$

$\Rightarrow$ In total $O(\log k + 2 \log \log k) \in O(\log k) + o(\log k)$

very good!

3p

Exercise 4 Random shuffle *6 Points*

Below are three different algorithms that are each supposed to produce a permutation of an array $a[1], \ldots, a[n]$ *uniformly at random* (we want each permutation to be equally likely). Which of the three algorithms are correct? Motivate your answer. We assume that **RAND(i,j)** produces an integer uniformly at random from $i, \ldots, j$, and that **SWAP(i,j)** exchanges $a[i]$ and $a[j]$.

```
SHUFFLE1:
for i from 1 to n do
  j <- RAND(i,n)
  SWAP(i,j)

SHUFFLE2:
for i from 1 to n do
  j <- RAND(1,n)
  SWAP(i,j)

SHUFFLE3:
for i from 1 to n do
  j <- RAND(1,i)
  SWAP(i,j)
```

SHUFFLE1: correct.

for $i=1$ n possibilitys for j $\Rightarrow$ n different permutations with each $\frac{1}{n}$ possibility to occur

for $i=2$ $n-1$ possibilitys for j $\Rightarrow$ $n-1$ different permutations with each $\frac{1}{n-1}$ possibility to occur

⋮

for $i=n$ 1 possibility for j

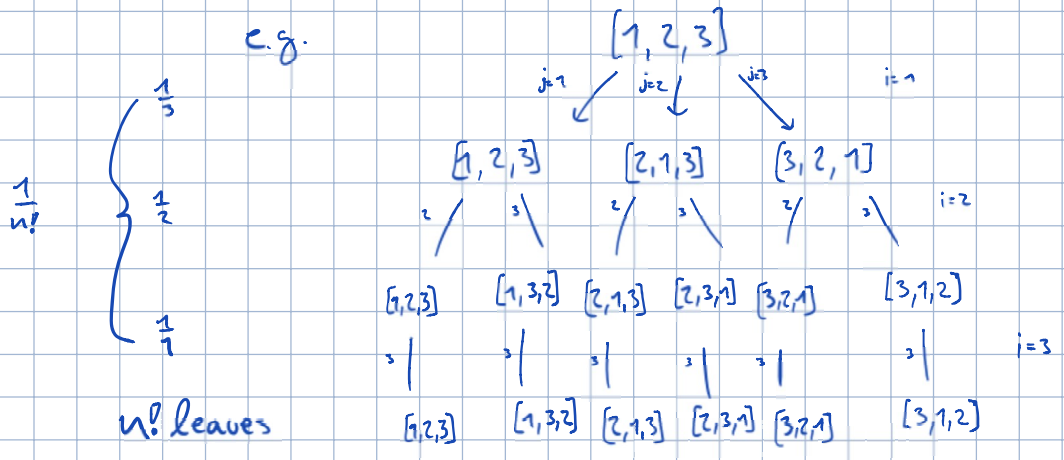
$\Rightarrow n \cdot (n-1) \cdot \ldots \cdot 1 = n!$

$\Rightarrow$ $n!$ possible permutations $\Rightarrow$ every possible permutation is occurable ✓

each permutation has $\frac{1}{n} \cdot \frac{1}{n-1} \cdot \ldots 1 = \frac{1}{n!}$ (the same) possibility to occur ✓

I'm doing this :)

e.g.



SHUFFLE 3: Correct

for $i=1$ 1 possibility for j $\Rightarrow$ 1 different permutations with each $\frac{1}{1}$ possibility to occur

for $i=2$ 2 possibilitys for j $\Rightarrow$ 2 different permutations with each $\frac{1}{2}$ possibility to occur

⋮

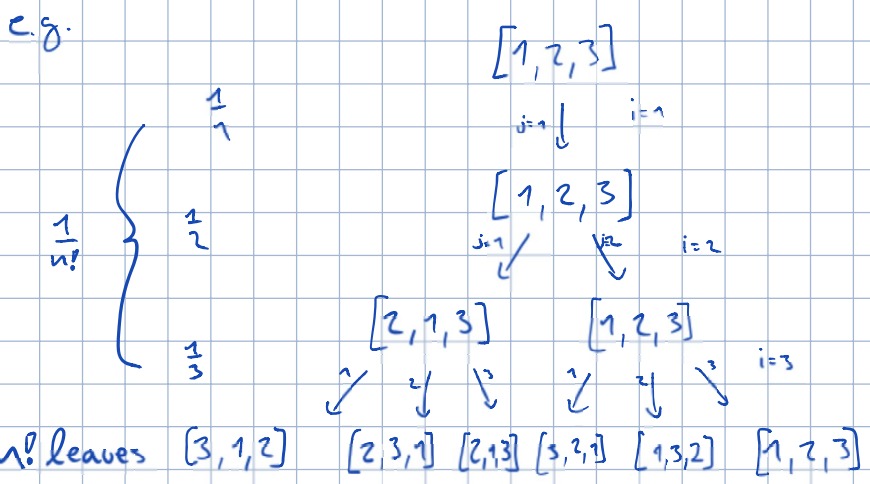
for $i=n$ n possibility for j $\Rightarrow$ n different permutations with each $\frac{1}{n}$ possibility to occur

$\Rightarrow$ $1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n = n_e^V$

$\Rightarrow$ $n!$ possible permutations $\Rightarrow$ every possible permutation is occurable ✓

each permutation has $\frac{1}{1} \cdot \frac{1}{2} \cdot \ldots \frac{1}{n} = \frac{1}{n!}$ (the same) possibility to occur ✓

e.g.



n! leaves [3,1,2] [2,3,1] [2,1,3] [3,2,1] [1,3,2] [1,2,3]

SHUFFLE 2: Correct

OSp

because $j \leftarrow$ rand$(1,n)$ has the same possibility for every number for the whole for-loop.

No, some elements moved again & again