CS 170 HW 1

2 (★★★) Analyze the running time

For each pseudo-code snippet below, give the asymptotic running time in Θ notation. Assume that basic arithmetic operations (+,−,×, and /) are constant time.

(a)

```
for i := 1 to n do
    j := 0;
    while j ≤ i do
    |   j := j + 2
    end
end
```

⇒ inner loop executes $\left(\frac{i}{2}\right)$ times

$\cancel{T(n) = \sum_{i=1}^{n} \left(\frac{i}{2}\right)^2}$

$\Theta(n^2)$

$T(n) = \sum_{i=1}^{n} \left(\frac{i}{2}\right)$

$= \frac{1}{2}(1+2+\dots+n) = \frac{1}{2}\left(\frac{n(n+1)}{2}\right)$

(b)

```
s := 0;
i := n;
while i ≥ 1 do
    i := i div 2;
    for j := 1 to i do
    |   s := s + 1
    end
end
```

⇒ while loop will execute $\cancel{\frac{n}{2}}$ log n times

⇒ for loop will execute i times

$\sum_{i=1}^{\log n} \left(\frac{n}{2^i}\right) = n \sum_{i=1}^{\log n} \left(\frac{1}{2}\right)^i = n \frac{1-\left(\frac{1}{2}\right)^{\log n}}{1-\frac{1}{2}}$

↳ for loop

↳ while loop

$= 2n\left(1-\left(\frac{1}{2}\right)^{\log n}\right)$

$= 2n - 2n\left(\frac{1}{2}\right)^{\log n} = \Theta(n)$

(c)

```
i := 2;
while i ≤ n do
|   i := i^2
end
```

~~-let x be number of times the loop executes~~

~~-loop will stop when~~

~~$2^{2x} > n$~~

~~$2x \log_2 2 > \log n$~~

~~$x > \frac{1}{2}\log n$~~

~~$\Theta(\log n)$~~

$\Theta(\log \log n)$

(d)

```
for i := 1 to n do
    j := i^2;
    while j ≤ n do
    |   j := j + 1
    end
end
```

⇒ for loop will execute n times

⇒ while loop will execute $(n-i^2)$ times

$\sum_{i=1}^{n} (n-i^2)$

→ note that $n \geq i^2$, otherwise while loop won't execute

→ so while loop will stop executing when $i > \sqrt{n}$

$\sum_{i=1}^{\sqrt{n}} (n-i^2) = \sum_{i=1}^{\sqrt{n}} n - \sum_{i=1}^{\sqrt{n}} i^2 = n\cdot\sqrt{n} - \frac{1}{3}n\sqrt{n}$

$= \Theta(n\sqrt{n})$

⇒ when $i \leq \sqrt{n}$, the while loop runs $(n-i^2)$ times

- otherwise it stops immidiately

$(n-\sqrt{n}) + \sum_{i=1}^{\sqrt{n}} (n-i^2)$

CORRECTIONS

2a

```
for i=1 to n do:
    j=0
    while j≤i do:
        j=j+2
```

$\Rightarrow$ the inner loop executes $\frac{i}{2}$ times

$$T(n) = \underbrace{\sum_{i=1}^{n}}_{\text{the for loop}} \overbrace{\left(\frac{i}{2}\right)}^{\text{while loop}} = \frac{1}{2}(1+2+\ldots+n)$$

$$= \frac{1}{2}\left(\frac{n(n+1)}{2}\right) = \frac{n^2}{4} + \frac{n}{4}$$

$\Rightarrow$ therefore

$$\Theta(n^2)$$

2b

```
s=0
i=n
while i≥1 do:
    i=i/2
    for j=1 to i do:
        s=s+1
```

$\Rightarrow$ while loop will execute $\log n$ times

- $\log n$ is the number of times it is possible to divide n by 2

$\Rightarrow$ for loop will execute ith times

- during k^{th} iteration $i = \frac{n}{2^k}$

$\Rightarrow$ therefore

$$\underbrace{\sum_{i=1}^{\log n}}_{\text{while loop}} \left(\frac{n}{2^i}\right) = n \sum_{i=1}^{\log n} \left(\frac{1}{2}\right)^i = n\left(\frac{1-\left(\frac{1}{2}\right)^{\log n}}{1-\frac{1}{2}}\right) = 2n\left(1-\left(\frac{1}{2}\right)^{\log n}\right)$$

$$= \underline{\underline{\Theta(n)}}$$

2c

```
i=2
while i≤n do:
    i=i^2
```

$\Rightarrow$ since $i=2$, when we repeatly square it will be $2^{2\cdot 2\cdot \ldots \cdot 2}$

$\Rightarrow$ in k^{th} iteration it will be 2^{2^k}

because

iteration	i
0	2
1	2^2
2	$2^4 = 2^{2^2}$
3	$2^8 = 2^{2^3}$
4	$2^{16} = 2^{2^4}$

$\Rightarrow$ while loop will stop when $i > n$

$$\left.\begin{aligned} 2^{2^k} &> n \\ 2^k \log 2 &> \log n \\ 2^k &> \log n \\ k \log 2 &> \log \log n \\ k &> \log \log n \end{aligned}\right\}$$

$\Rightarrow$ while loop will stop after $k^{th} = \log \log n$ iteration

$$\underline{\underline{\Theta(\log \log n)}}$$

CORRECTIONS

2d

```
for i=1 to n do:
    j=i^2
    while j<=n do:
        j=j+1
```

$\Rightarrow$ for loop will run n times

$\Rightarrow$ when $i \leq \sqrt{n}$, while loop will run $(n-i^2)$ times

$\Rightarrow$ when $i > \sqrt{n}$, while loop won't run

$$(n-\sqrt{n}) + \sum_{i=1}^{\sqrt{n}} (n-i^2) = n-\sqrt{n} + \sum_{i=1}^{\sqrt{n}} - \sum_{i=1}^{\sqrt{n}} i^2$$

$$= n-\sqrt{n} + n\sqrt{n} - \frac{1}{3}(\sqrt{n})^3$$

$$= n-\sqrt{n} + n\sqrt{n} - \frac{1}{3}n\sqrt{n} = \underline{\underline{\Theta(n\sqrt{n})}}$$

Note that

$$\sum_{i=1}^{n} i^2 \approx \frac{1}{3}n^3$$

3 (★★★) Asymptotic Complexity Comparisons

(a) Order the following functions so that $f_i = O(f_j) \iff i \leq j$. Do not justify your answers.

NOTE

$f_3 < f_7 < f_2 < f_5 < f_4 < f_9 < f_8 < f_6 < f_1$

$12 < \log_2 n < n^{\frac{1}{3}} < \sqrt{n} < n < n^3 < 2^{\sqrt{n}} < 2^n < 3^n$

(i) $f_1(n) = 3^n$

(ii) $f_2(n) = n^{\frac{1}{3}}$

(iii) $f_3(n) = 12$

(iv) $f_4(n) = 2^{\log_2 n} = n^{\log_2 2} = n$

(v) $f_5(n) = \sqrt{n}$

(vi) $f_6(n) = 2^n$

(vii) $f_7(n) = \log_2 n$

(viii) $f_8(n) = 2^{\sqrt{n}}$

(ix) $f_9(n) = n^3$

b) In each of the following, indicate whether $f=O(g)$, $f=\Omega(g)$ or both (in which case $f=\Theta(g)$). Briefly explain each of your answers.

Racall that in asymptotic growth rate
logarithmic < linear < polynomial < exponential

	$f(n)$	$g(n)$	
i) ✓	$\log_3(n)$ $f=\Theta(g)$	$\log_4(n)$	because they are both logarithmic functions (use formula $f=\frac{\log n}{\log 3}$, $g=\frac{\log n}{\log 4}$ $\Leftrightarrow$ differ only by constant factor
ii) ✓	$n\log(n^4)$ $f=O(g)$	$n^2\log(n^3)$	$f=n\log(n^4)=4n\log n=\Theta(n\log(n))$ $g=n^2\log(n^3)=3n^2\log n=\Theta(n^2\log(n))$
iii) ✓	$\sqrt{n}$ $f=\Omega(g)$	$(\log n)^3$	$\lim_{n\to\infty}\frac{n^{\frac{1}{2}}}{(\log n)^3}=\frac{\frac{1}{2}n^{-\frac{1}{2}}}{3(\log n)^2\cdot\frac{1}{n}}=k\cdot\frac{n^{\frac{1}{2}}}{(\log n)^2}$ $=k\frac{n^{\frac{1}{2}}}{\log n}=\lim_{n\to\infty}n^{\frac{1}{2}}$ Any polynomial dominates product of logs
iv) ✓	2^n $f=\Theta(g)$	2^{n+1}	$f=2^n=\Theta(2^n)$ $g=2^{n+1}=2\cdot 2^n=\Theta(2^n)$
v) ⊗	n $f=\Omega(g)$	$(\log n)^{\log\log n}$	$n=2^{\log n}$ $(\log n)^{\log(\log n)}=2^{(\log(\log n))^2}$ $\to f$ grows faster than g since $\log n$ grows faster than $(\log(\log n))^2$
vi) ✓	$n+\log n$ $f=\Theta(g)$	$n+(\log n)^2$	$\lim_{n\to\infty}\left(\frac{n+\log n}{n+(\log n)^2}\right)=\lim_{n\to\infty}\left(\frac{1+\frac{1}{n}}{1+2\log n\cdot\frac{1}{n}}\right)$ $=\frac{\frac{n+1}{n}}{\frac{n+2\log n}{n}}=\frac{n+1}{n+2\log n}=\frac{1}{1+\frac{1}{n}}=\frac{1}{\frac{n+1}{n}}$ $=\lim_{n\to\infty}\frac{n}{n+2}=1$ $\to$ linear term dominates
vii) ⊗	$\log(n!)$ ~~$f=O(g)$~~ $f=\Theta(g)$	$n\log n$	Note $n\log n=\log(n^n)$ Since n^n grows faster than $n!$ $\log(n!)=O(\log n^n)$

CORRECTIONS

3b (viii) $f = \log(n!)$

$g = n \cdot \log n = \log(n^n)$

$$n! \leq 1 \cdot 2 \cdot \ldots (n-1) \cdot n \leq n \cdot n \cdot \ldots n \leq n^n$$

=> WLOG assume that n is even

$$n! = 1 \cdot 2 \cdot \ldots \cdot n \geq n \cdot (n-1) \cdot \ldots (n - \tfrac{n}{2}) \geq \left(\tfrac{n}{2}\right)^{\frac{n}{2}+1}$$

=> therefore

$$\left(\tfrac{n}{2}\right)^{\frac{n}{2}} \leq n! < n^n$$

=> then

$$\tfrac{n}{2} \log\left(\tfrac{n}{2}\right) \leq \log(n!) \leq n \log n$$

-> therefore function grow at the same rate

$$f = \Theta(g)$$

CORRECTIONS

3b (v) $f = n$

$g = (\log n)^{\log \log n}$

Note $n = 2^{\log n}$

$(\log n)^{\log \log n} = 2^{(\log \log n)^2}$

O

ASK

$\Rightarrow$ so f grows faster than g since $\log n$ grows faster than $(\log \log n)^2$

✓

4 (★★) Bit Counter

Consider an n-bit counter that counts from 0 to 2^n.

When $n = 5$, the counter has the following values:

Step	Value	# Bit-Flips
0	00000	–
1	00001	1
2	00010	2
3	00011	1
4	00100	3
⋮	⋮	
31	11111	1
31	00000	5

Alternative solution:
- number of times i^{th} bit from left is flipped is 2^i
  - first bit flipped once
  - last bit flipped every time : 2^n

$\Rightarrow$ total number of flips

$\sum_{n=1}^{n} 2^i = 2^{n+1} - 2 = \Theta(2^n)$

For example, the last two bits flip when the counter goes from 1 to 2. Using $\Theta(\cdot)$ notation, find the growth of the *total* number of bit flips (the sum of all the numbers in the "# Bit-Flips" column) as a function of n.

n=1

0	–
1	1
0	1

2

n=2

00	–
01	1
10	2
11	1
00	2

6

n=3

000	–
001	1
010	2
011	1
100	3=n
101	1
110	2
111	1
000	3=n

14

T(3)

n=4

0000	–
0001	1
0010	2
0011	1
0100	3
0101	1
0110	2
0111	1
1000	4=n
1001	1
1010	2
1011	1
1100	3
1101	1
1110	2
1111	1
0000	4=n

30

$T(4) = 2 \cdot T(3) - 2(4-1) + 2(4)$

n=5

00000	–
00001	1
⋮	
01111	1
10000	5
10001	1
⋮	
11111	1
00000	5

$T(n) = 2 \cdot T(n-1) - 2(n-1) + 2 \cdot n$

$= 2 \cdot T(n-1) - 2n + 2 + 2n$

$= 2 \cdot T(n-1) + 2$

$\Rightarrow$ Recurrence Relation for the problem

$T(n) = 2 \cdot T(n-1) + 2$

$= 2(2T(n-2) + 2) + 2$

1 problem	of size n at cost	2	2
2	$n-1$	2	$2 \cdot 2$
2^i	$n-i$	2	$2^i \cdot 2$

$\rightarrow$ stops when $i = n$

$T(n) = \sum_{i=1}^{n} 2^i \cdot 2 = 2 \sum_{i=1}^{n} 2^i = 2\,\frac{1-2^n}{1-2}$

$= -2(1-2^n) = 2 \cdot 2^n - 2 = 2^{n+1} - 2$

$\Theta(2^n)$

5 (★★) Recurrence Relations

✓ (a) $T(n) = 4T(n/2) + 42n$

$a = 4$ $\quad \frac{a}{b^d} = \frac{4}{2^1} = 2 > 1$

$b = 2$

$d = 1$ $\quad O(n^{\log_b a})$

By master theorem $\quad O(n^{\log_2 4}) = \underline{\underline{O(n^2)}}$

✓ (b) $T(n) = 4T(n/3) + n^2$

$a = 4$ $\quad \frac{a}{b^d} = \frac{4}{3^2} = \frac{4}{9} < 1$

$b = 3$

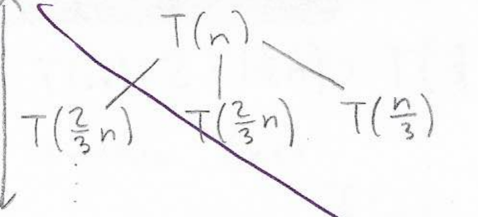
$d = 2$ $\quad O(n^d)$

By master theorem $\quad \underline{\underline{O(n^2)}}$

$i=1$

(c) $T(n) = 2T(2n/3) + T(n/3) + n^2$

$h=\log_3 n$



1 problem of size	n	cost n^2	n^2
2	$\frac{2}{3}n$	$(\frac{2}{3}n)^2$	$2\cdot(\frac{2}{3}n)^2 = 2(\frac{2}{3})^2 n^2$
1	$\frac{1}{3}n$	$(\frac{1}{3}n)^2$	$(\frac{1}{3})^2 n^2$
2^i	$(\frac{2}{3})^i n$	$((\frac{2}{3})^i n)^2$	$2^i(\frac{2}{3})^{2i} n^2$
1	$(\frac{1}{3})^i n$	$((\frac{1}{3})^i n)^2$	$(\frac{1}{3})^{2i} n^2$

$$T(n) = n^2 + \left[2^1\left(\tfrac{2}{3}\right)^2 n^2 + \dots + 2^i\left(\tfrac{2}{3}\right)^{2i} n^2 + \dots + 2^{\log_3 n}\left(\tfrac{2}{3}\right)^{2\log_3 n} n^2\right] + \left[\left(\tfrac{1}{3}\right)^2 n^2 + \dots + \left(\tfrac{1}{3}\right)^{2i} n^2 + \dots + \left(\tfrac{1}{3}\right)^{2\log_3 n} n^2\right] = n^2 + \sum_{i=1}^{\log_3 n} 2^i \left(\tfrac{2}{3}\right)^{2i} n^2 + \sum_{i=1}^{\log_3 n}\left(\tfrac{1}{3}\right)^{2i} n^2$$

$$= n^2 + n^2 \sum_{i=1}^{\log_3 n} 2^i\left(\tfrac{2}{3}\right)^{2i} + n^2 \sum_{i=1}^{\log_3 n}\left(\tfrac{1}{3}\right)^{2i}$$

$$= n^2 + n^2 \sum_{i=1}^{\log_3 n} \frac{(2^{\frac{3}{2}})^{2i}}{3^{2i}} + n^2 \sum_{i=1}^{\log_3 n}\left(\tfrac{1}{3}\right)^{2i} = n^2 + n^2 \sum_{i=1}^{\log_3 n}\left(\frac{2{,}83}{3}\right)^{2i} + n^2 \sum_{i=1}^{\log_3 n}\left(\tfrac{1}{3}\right)^{2i}$$

WRONG
-on extra sheet

$\Rightarrow$ as n goes to infinity both sums goes to some constants

$$T(n) = n^2 + k_1 n^2 + k_2 n^2 = \boxed{\Theta(n^2)}$$

(d) $T(n) = 3T(n/4) + n\log n$

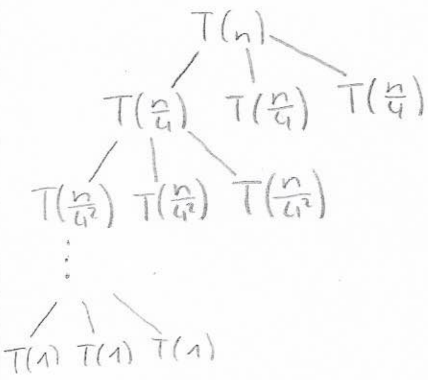
✓

1 problem of size	n	at cost $n \log n$	$n \log n$
3	$\frac{n}{4}$	$\frac{n}{4}\log(\frac{n}{4})$	$3\frac{n}{4}\log(\frac{n}{4})$
3^i	$\frac{n}{4^i}$	$\frac{n}{4^i}\log(\frac{n}{4^i})$	$3^i\frac{n}{4^i}\log(\frac{n}{4^i})$

$$T(n) = n\cdot\log n + 3\cdot\frac{n}{4}\log\left(\frac{n}{4}\right) + \dots + 3^i\frac{n}{4^i}\log\left(\frac{n}{4^i}\right) + \dots\ 3^{\log_4 n}\cdot\frac{n}{4^{\log_4 n}}\cdot\log\left(\frac{n}{4^{\log_4 n}}\right)$$

$$= n\left[\log n + \frac{3}{4}\log\left(\frac{n}{4}\right) + \dots + \left(\frac{3}{4}\right)^i\log\left(\frac{n}{4^i}\right) + \dots + \left(\frac{3}{4}\right)^{\log_4 n}\log\left(\frac{n}{4^{\log_4 n}}\right)\right]$$

$$= n\sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i \log\left(\frac{n}{4^i}\right) = n\sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i\left[\log n - \log 4^i\right]$$



$$= n\sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i\left[\log n - i\log 4\right]$$

$$= n\log n\sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i - n\log 4\sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i i$$

$h=\log_4 n$

$$= n\log n\cdot\left(\frac{1-\left(\frac{3}{4}\right)^{\log_4 n}}{1-\frac{3}{4}} + 1\right) - n\log 4\sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i\cdot i$$

$\Rightarrow$ as $n\to\infty$, $\left(\frac{1-\frac{3}{4}^{\log_4 n}}{\frac{1}{4}} + 1\right)$ goes to 1

and $\sum_{i=0}^{\log_4 n}\left(\frac{3}{4}\right)^i\cdot i$ goes to 12

$\Rightarrow$ therefore

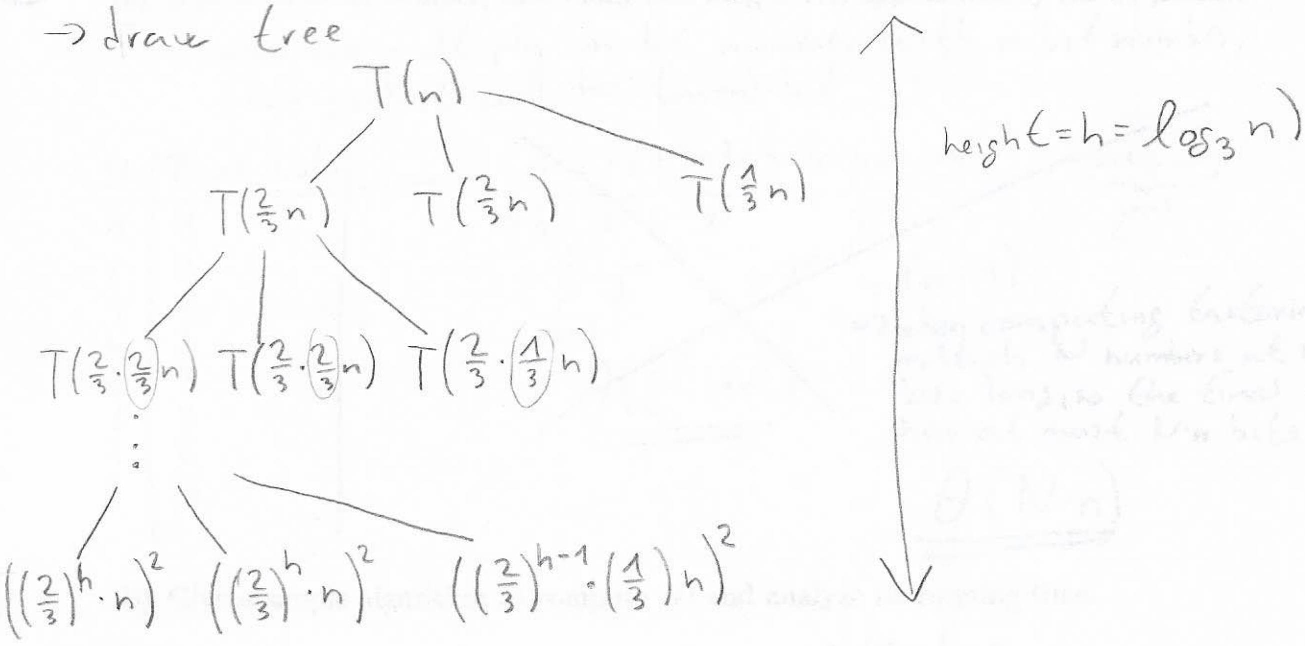
$$T(n) = n\log n\cdot 1 - n\log 4\cdot 12 = n\log n - cn$$
$$= n\log n$$

$\boxed{\Theta(n\log n)}$

CORRECTION

5c $T(n) = 2T\left(\frac{2}{3}n\right) + T\left(\frac{1}{3}n\right) + n^2$

→ draw tree



- since height of tree is $\log_3 n$ and every node takes n^2 time to compute

$$n^2 \cdot \log_3 n = \boxed{\Theta(n^2 \log n)}$$

6 (★★) Computing Factorials

Consider the problem of computing $N! = 1 \times 2 \times \cdots \times N$.

(a) If N is an n-bit number, how many bits long is $N!$, approximately (in $\Theta(\cdot)$form)?

=> when we multiply m-bit number with n-bit number,
their product will be (m+n)-bit

N	# of bits
1	1
2	2
3	2
4	3
5	3
6	3
7	3
8	4

$$1+\underbrace{2+2}_{2}+\underbrace{3+3+3+3}_{2^2}+\underbrace{4+\ldots+4}_{2^3}+\ldots+\underbrace{n+\ldots+n}_{2^{n-1}} \qquad 2^0=1$$

$$= \sum_{i=1}^{n} 2^{n-1} = 2^{n-1} \cdot n(n+1)$$

$$\Theta(2^n \cdot n^2)$$

=> when computing factorial, we multiply N numbers at most N bits long, so the final number has at most N·n bits

$$\Theta(N \cdot n)$$

(b) Give a simple algorithm to compute $N!$ and analyze its running time.

• recursive

```
factorial(N)
  if N≤1:
    return 1
  else
    return N·factorial(N-1)
```

• iterative

```
factorial(N)
  result=1
  for I=2 to N:
    result=result·I
  return result
```

=> analysis for iterative

-> for loop will run N times

-> multiplication of result · I will cost $(N\cdot i)\cdot i = N \cdot i^2$

result: $N \cdot i$ bit number → $N\cdot n$-bit number maximum

I: i-bit number → n-bit number maximum

=> multiplication will cost $N \cdot n^2$ maximum every for loop iteration

-> N for loop iterations will be

$$N \cdot (N \cdot n^2) = N^2 \cdot n^2$$

$$\Theta(N^2 \cdot n^2)$$

7 (★★★) Four-subpart Algorithm Practice

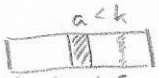
Given a sorted array A of n integers, you want to find the index at which a given integer k occurs, i.e. index i for which $A[i] = k$. Design an efficient algorithm to find this i.

Main idea:

✓

$\Rightarrow$ perform binary search
- $\rightarrow$ pick element a in the middle of A
- $\rightarrow$ if $a=k$, return index of a
- $\rightarrow$ if $a<k$, perform binary search on right half of A
- $\rightarrow$ if $a>k$, perform binary search on left half

$a<k$



Psuedocode:

✓

```
search(sorted array A, size of A = n, k)
                                  └> A[1,...,n]
    a = A[n/2]
    i = n/2
    if n≤0: return -1      # in case integer k is not in A, return -1
    if a<k: return search(A[n/2+1, n], n - n/2, k)
    else if a>k: return search(~~A[0, n/2]~~, n/2, k)
                               A[0, n/2 - 1]
    else: return i
```

$\Rightarrow$ we can also do it without passing size of array
- just change if $a<k$: return i + search($A[\frac{n}{2}+1, n]$, k)

Proof of correctness:

⊗

~~$\Rightarrow$ when we pick element a in the middle of sorted array A, there are 3 possibilities~~
- ~~① $a=k$ - we found a is right element and we can return its index~~
- ~~② $a<k$ - the right element has to be in right half of sorted array~~
- ~~③ $a>k$ - the right element has to be in left half of sorted array~~

~~- during every iteration, we must check if size of array is equal to 0~~
- ~~when $n \le 0$, it means a wasn't found in A~~

$\Rightarrow$ proof by induction

Running time analysis:

✓

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

comparison

$a = 1$
$b = 2$
$d = 0$

$\Rightarrow$ use master theorem

$$\frac{a}{b^d} = \frac{1}{2^0} = 1 > 0 \qquad \Theta(n^d \log n) = \underline{\Theta(\log n)}$$

CORRECTIONS

7: proof of correctness

=> proof by induction
- if array of size n contains k, binary search will find it

base case

if $n=1$, then $A[1]=k$ because k is in A

inductive hypothesis
- binary search works on arrays of size $\leq m$ for some m
  - correct index is returned when present

inductive step
- prove it for array of size $(m+1)$
- if $A[\frac{m}{2}]=k$, output correct index
- else we recurse on one half of A
  - because A is sorted, our comparison ensures that we recurse on the half of A that contains k
    - -> recursive call will be correct by induction hypothesis
      - because one half of array has size $\leq m$
    - -> by induction, we will find correct index
- if k is not present, the algorithm will not return valid index

8 (★★★) Hadamard matrices

The Hadamard matrices $H_0, H_1, H_2, \dots$ are defined as follows:

- H_0 is the 1×1 matrix $[1]$
- For $k > 0$, H_k is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

(a) Write down the Hadamard matrices H_0, H_1, and H_2.

$\Rightarrow H_0$ is 1×1 matrix

$$H_0 = [1]$$

$\Rightarrow H_1$ is 2×2 matrix

$$H_1 = \begin{bmatrix} H_0 & H_0 \\ H_0 & -H_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

$\Rightarrow H_2$ is 4×4 matrix

$$H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

(b) Compute the matrix-vector product $H_2 v$, where H_2 is the Hadamard matrix you found above, and $v = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$ is a column vector. Note that since H_2 is a 4×4 matrix, and v is a vector of length 4, the result will be a vector of length 4.

$$H_2 \cdot v = \underset{\substack{4\times4 \\ r\ c}}{\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}} \cdot \underset{\substack{4\times1 \\ r}}{\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}} = \underset{4\times1}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}}$$

(c) Now, we will compute another quantity. Take v_1 and v_2 to be the top and bottom halves of v respectively. Therefore, we have that $v_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, and $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$. Compute $u_1 = H_1(v_1 + v_2)$ and $u_2 = H_1(v_1 - v_2)$ to get two vectors of length 2. Stack u_1 above u_2 to get a vector u of length 4. What do you notice about u?

$$u_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = \underset{2\times 2}{\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}} \cdot \underset{2\times 1}{\begin{bmatrix} 0 \\ 0 \end{bmatrix}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$u_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix} - \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix} = H_2 \cdot v$$

=> I noticed that we got the same vector as when we were multiplying H_2 and v

(d) Suppose that

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

is a column vector of length $n = 2^k$. v_1 and v_2 are the top and bottom half of the vector, respectively. Therefore, they are each vectors of length $\frac{n}{2} = 2^{k-1}$. Write the matrix-vector product $H_k v$ in terms of H_{k-1}, v_1, and v_2 (note that H_{k-1} is a matrix of dimension $\frac{n}{2} \times \frac{n}{2}$, or $2^{k-1} \times 2^{k-1}$). Since H_k is a $n \times n$ matrix, and v is a vector of length n, the result will be a vector of length n.

$$H_k \cdot v = u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} H_{k-1}(v_1 + v_2) \\ H_{k-1}(v_1 - v_2) \end{bmatrix}$$

$u_1 = H_{k-1}(v_1 + v_2)$

$u_2 = H_{k-1}(v_1 - v_2)$

(e) Use your results from (c) to come up with a divide-and-conquer algorithm to calculate the matrix-vector product $H_k v$, and show that it can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time.

multiply (matrix H_k, size of matrix k, vector v):
  H_{k-1} = first 2^{k-1} rows and 2^{k-1} columns of matrix H_k
  v_1 = upper 2^{k-1} rows of v
  v_2 = bottom 2^{k-1} rows of v

  if $k=1$ — base case
    $u_1 = H_{k-1}(v_1+v_2)$
    $u_2 = H_{k-1}(v_1-v_2)$
    $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$
    return u
  else:
    a = multiply $(H_{k-1}, k-1, v_1)$ # $= H_{k-1} \cdot v_1$
    b = multiply $(H_{k-1}, k-1, v_1)$ # $= H_{k-1} \cdot v_2$
    $u_1 = a+b$
    $u_1 = a-b$
    $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$
    return u

$T(n) = 2T(\frac{n}{2}) + \cancel{\times} O(n)$

- we need to find vectors (v_1+v_2) and (v_1-v_2), which takes $O(n)$
- we also need to find $H_{k-1}(v_1+v_2)$ and $H_{k-1}(v_1-v_2)$ which takes $T(\frac{n}{2})$ time

$a=2$
$b=2$
$d=\cancel{0}\,1$

~~$\frac{a}{b^d} = \frac{2}{2^0} = 2 > 1$~~

$\frac{a}{b^d} = \frac{2}{2^1} = 1$

$O(n^d \log n)$

$O(n \log n)$