Nama : Alvaro Austin
NPM: 2106752180
Kelas : A
Kode Asdos : 3

Dengan ini saya menyatakan bahwa PR ini adalah hasil Pekerjaan saya sendiri

(1) Soal 1 : Gunakan master theorem untuk memperoleh tight asymptotic bound untuk persamaan rekurensi berikut :

a) $T(n) = T(11n/12) + n^2$

Berdasarkan informasi diatas kita memperoleh bahwa $a = 1$, $b = \frac{12}{11}$ dan $f(n) = n^2$

↳ kita juga bisa mencari $t = {}^{\frac{12}{11}}\log(1) = 0$

Oleh karena itu dibandingkan $f(n)$ dan n^t

Bandingkan $f(n)$ dengan $n^{\log_b a} = n^t$

$n^2 = n^p$ dengan $n^{\frac{12}{11}\log 1} = n^0$

(1) Kita dapat memilih case 3 dimana $p > t$ dengan extra rule. $af(\frac{n}{b}) < kf(n)$ dengan $k < 1$ terpenuhi

Regularity condition:

(2) $f\left(\frac{11n}{12}\right) < kf(n) \Rightarrow \left(\frac{11n}{12}\right)^2 < kn^2$

$\frac{11^2}{12^2} n^2 < kn^2$

→ terbukti karena k bernilai $\frac{11^2}{12^2}$ dimana pasti k kurang dari 1 sehingga terbukti bahwa $T(n) = \Theta(n^2)$

$f(n) = \Omega(n^s)$ dimana $s > t$, contoh ambil $s = 1$ bahwa $n^2 = \Omega(n)$ terpenuhi

Step ini terlebih dahulu

maka $T(n) = \Theta(f(n))$

atau $T(n) = \Theta(n^2)$

Namun perlu juga dibuktikan regularity condition terpenuhi / tidak

Case 3 terpenuhi: Kesimpulan karena nilai polinomial p dari n^2 lebih besar jadi $T(n) = \Theta(f(n))$
$= \Theta(n^2)$

b) $T(n) = 8T\left(\frac{n}{2}\right) + \frac{n^3}{\lg(n)}$

Berdasarkan informasi diatas kita dapat melihat bahwa $a = 8$ $b = 2$ → $t = \log_b a = \log_2 8 = 3$

$f(n) = \frac{n^3}{\lg(n)}$ → p

$f(n) = n^3 \cdot \lg^{-1}(n)$

Karena nilai $p = t$ maka kita dapat memilih case 2 : Terdapat 2 kasus bagi case 2

↳ weak version: kasus ini tidak memenuhi karena perbedaan $f(n)$ dengan n^t

$\frac{n^3}{\log(n)} : n^3$ tapi tidak polynomially the same

↳ strong version : kita dapat define $f(n)$ sebagai $f(n) = n^3 \cdot \log^{-1}(n)$
small function of n

Namun strong version memiliki aturan untuk $f(n) = n^p \cdot \log^k(n)$ untuk $k \geq 0$, akan tetapi untuk kasus ini $f(n) = n^3 \cdot \log^{-1}(n)$ dimana $k = -1$. Hal ini artinya strong version juga tidak memenuhi karena nilai k dari log yg bernilai negatif.

Oleh karena itu, karena nilai k tidak lebih besar sama dengan 0, maka strong version tidak memenuhi.

Kesimpulan: case 2 tidak dapat diaplikasikan baik itu weak maupun strong version //

Sehingga tidak berlaku master theorem pada kasus ini.

Nama : Alvaro A
NPM : 2106952180
Kelas : A
Kode A : 3

(c) $T(n) = 9T(\frac{n}{3}) + n \lg n$

Berdasarkan pertanyaan diatas, kita memperoleh bahwa $a = 9$, $b = 3 \rightarrow t = \log_b a = \log_3 9 = 2$.

$F(n) = n \cdot \lg n$

Maka dibandingkan $F(n)$ dengan n^t

$F(n) = n \lg n$
↳ $p = 1$

$n^t = n^{\log_b a}$
$n^t = n^{\log_3 9}$
$n^t = n^2$
$t = 2$

Dapat dilihat bahwa $p < t$, kita bisa menggunakan case 1 :
Apabila $F(n) = O(n^s)$ dimana $s < t$ maka $T(n) = \Theta(n^t)$

↳ Mari kita pilih $s = \frac{3}{2}$ dimana $s < t = s < 2$

Dapat dilihat bahwa untuk membuktikan $F(n) = O(n^{\frac{3}{2}})$
$n \lg n \leq c \cdot n^{\frac{3}{2}}$, mari pilih $c = 1$
$n \cdot \lg n \leq n^{\frac{3}{2}}$, kita tahu bahwa ini berlaku untuk $n \geq 1$

Sehingga karena terbukti bahwa
$n \lg n = O(n^{\frac{3}{2}})$ maka terbukti
juga bahwa $T(n) = \Theta(n^2)$ (case 1 terpenuhi)

(d) $T(n) = 16T(\frac{n}{4}) + 2^{\lg(n+1)^2}$

Berdasarkan pernyataan diatas kita dapat memperoleh nilai $a = 16$, $b = 4 \rightarrow$ dimana $t = \log_4 16 = 2$

$F(n) = 2^{\lg(n+1)^2}$
$= (n+1)^2$ karena sifat $a^{^a\log b} = b$

Menggunakan case 2 definisi formal, kita mengetahui bahwa case 2 :
$F(n) = \Theta(n^{\log_b a} \log^k n)$ untuk $k \geq 0$

dimana kita memperoleh
$(n+1)^2 = \Theta(n^{\log_4 16} \log^k(n))$
$(n+1)^2 = \Theta(n^2 \log^0(1)) \rightarrow k = 1$
$(n+1)^2 = \Theta(n^2)$

kita perlu membuktikan bahwa $(n+1)^2 = \Theta(n^2)$ ini berlaku, apabila berlaku maka $T(n) = \Theta(n^2 \log n)$

Buktikan dengan : limit : $0 < \lim_{n\to\infty} \left(\frac{F(n)}{g(n)}\right) < \infty$ maka $F(n) = \Theta(g(n))$

$\lim_{n\to\infty} \frac{(n+1)^2}{n^2} \rightarrow \lim_{n\to\infty} \frac{n^2+2n+1}{n^2} = \lim_{n\to\infty} 1 + \frac{2}{n} + \frac{1}{n^2} = 1$

maka dapat dilihat bahwa $\lim \frac{(n+1)^2}{n^2} = 1$ dimana memenuhi karena berada diantara 0 dan ∞
maka terbukti bahwa : $(n+1)^2 = \Theta(n^2)$

Karena terbukti bahwa $(n+1)^2 = \Theta(n^2)$, kita juga dapat mengatakan bahwa $T(n) = \Theta(n^2 \log n)$ sehingga case 2 terpenuhi

Nama : Alvaro A
NPM : 2106732180
Kelas : A
Kode Asisten : 3

1) Asumsi : level 1 = root of the node
Base case : $T(1) = 1$

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

a) Kita dapat memperoleh level recursion tree dari melihat perluasan tree ini. Berdasarkan perluasan terbuntar.
Berdasarkan soal diatas kita dapat melihat bahwa $T\left(\frac{n}{2}\right)$ memiliki perluasan yg lambat maka, setiap level misal dan level 1 : $\frac{n}{2}$, level 2 : $\frac{n}{2^2}$, level 3 : $\frac{n}{2^3}$, ... level ke-(maksimum) : $\frac{n}{2^i}$

sehingga : kita tahu bahwa level i adalah base case dimana bernilai 1, maka

$1 = \frac{n}{2^i} \rightarrow 2^i = n$ → yg berarti $i = {}^2\log n$ (properti log $a^{\log_b c} = c$ ← $a^c = b$)

oleh karena itu kita memperoleh level maksimum sebesar $^2\log n + 1$ karena mulai dari 1 levelnya

b) Tree tersebut tidak penuh ke level maksimumnya. Level tersebut mulai ?, dan penuh ketika Penuh sampai

Cabang rekursi $T\left(\frac{n}{4}\right)$ sudah mencapai base case dan tidak lanjut lagi. Hal ini, karena tree termasuk penuh apabila node pada level tersebut berjumlah $3^{\text{(level)}}$. Maka pada saat Cabang rekursi $T\left(\frac{n}{4}\right)$ selesai, maka node pada level tersebut tidak berjumlah $3^{\text{(level)}}$.

Oleh karena itu, tree mulai tidak penuh pada saat lebih dari level $\left(\log_4(n) + 1\right)$ karena $\log_4(n)+1$ adalah level maks untuk $T\left(\frac{n}{4}\right)$, karena rekursi untuk $T\left(\frac{n}{4}\right)$ sudah terjadi

c) Tidak, setiap node dalam recurrence tree tidak memiliki cost yg sama

Cost masing2 node pada level 1, 2, 3 serta level :

Total

Cost level 1 = $\frac{n}{2}$ → $\frac{n}{2}$

level 2 = $\frac{n}{8}, \frac{n}{8}, \frac{n}{4}$ → $\frac{n}{8} + \frac{n}{8} + \frac{n}{4} = \frac{2n}{8} + \frac{n}{4} = \frac{2n}{4} = \frac{n}{2}$

level 3 = $\frac{n}{32}, \frac{n}{32}, \frac{n}{16}, \frac{n}{32}, \frac{n}{32}, \frac{n}{16}, \frac{n}{16}, \frac{n}{16}, \frac{n}{8}$ → $\frac{n}{2}$

Terlihat bahwa pada level 1, level 2, level 3, memiliki node yg berbeda-beda.
Akan tetapi total untuk ketiga level tersebut sama $\left(\frac{n}{2}\right)$

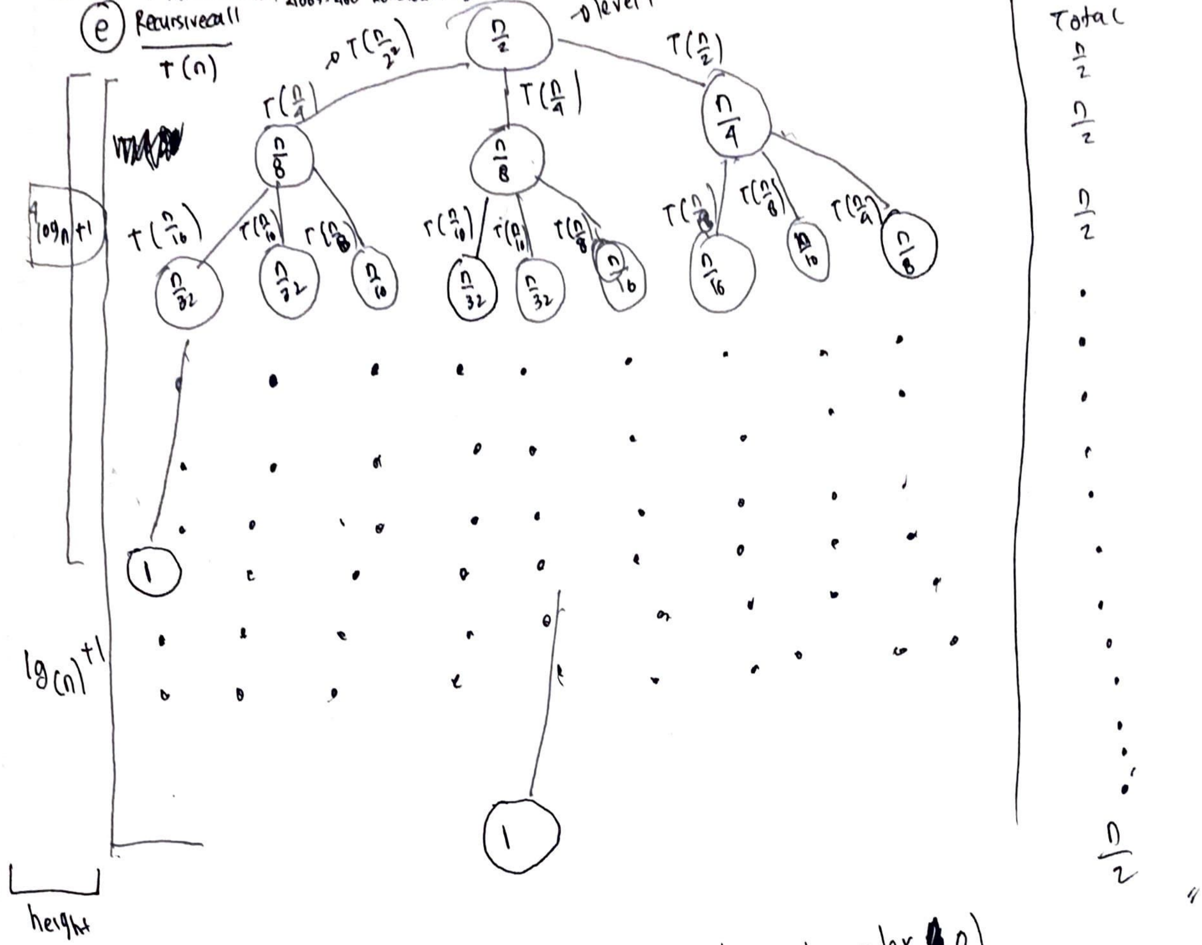
d) Upperbound $T(n)$:
Karena tinggi dari recursion tree $\leq \log_2(n)$ dan biaya setiap level adalah $\leq \frac{n}{2}$. Maka estimasi upperbound untuk rekurensi diatas adalah $O\left(\frac{n}{2} \cdot \log_2(n)\right)$
→ dihilangkan konstanta : $O(n \lg n)$

sehingga ini adalah upperboundnya

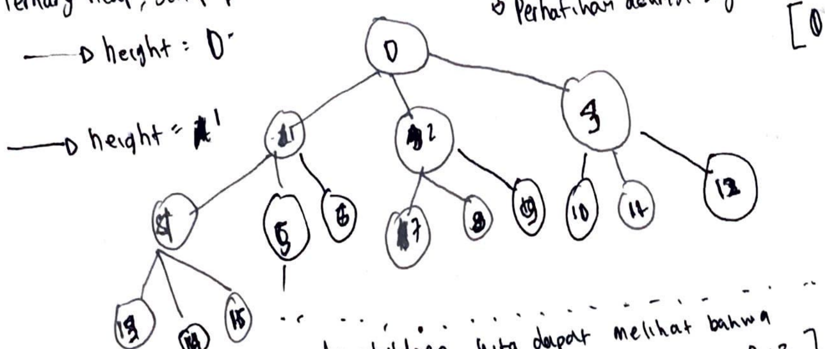
lower bound : $\Omega\left(n \cdot \log_4 n\right)$

Nama: Alvaro P Kelas: A NPM: 2106752180 Kode asdos: 3

(e) Recursive call



(3) a) Ternary Heap, setiap child memiliki 3 anak: (Asumsi array dimulai dari index 0)
Perhatikan asumsi saya: [0, 1, . . . n-1]



Untuk menentukan index parent dan children, kita dapat melihat bahwa

- Parent: $\lceil \frac{i}{3} \rceil - 1$
- Child kiri: $3i+1$
- Child tengah: $3i+2$
- Child kanan: $3i+3$

→ contoh: parent dari elemen 13: $\lceil \frac{13}{3} \rceil - 1 = 5 - 1 = 4$

child kiri 4: $3(4)+1 = 13$
kanan 4: $3(4)+3 = 15$
tengah 4: $3(4)+3 = 14$

(b) Akan dibuat algoritma TERNARY-MAX-HEAPIFY:

Nama: Alvaro Pustin
Kelas: A
NPM: 2106752180
Kode Asisten: 3

```
TERNARY-MAX-HEAPIFY(A, i):
    l = 3*i+1        # variable left child
    m = 3*i+2        # variable mid child
    r = 3*i+3        # variable right child
    # mencari index terbesar dari parent, left, middle, right, nodes
    min_i = i
    IF(l < len(A) and A[l] > A[min_i]):
        min_i = l
    IF(m < len(A) and A[m] > A[min_i]):
        min_i = m
    IF(r < len(A) and A[r] > A[min_i]):
        min_i = r
    IF(min_i != i):
        A[i], A[min_i] = A[min_i], A[i]
        TERNARY-MAX-HEAPIFY(A, min_i)
```

Komplesitas heapify bergantung dengan tinggi dari tree yg dibuat. Oleh karena itu, karena ternary max heapify memiliki tinggi $\log_3(n)$ maka komplesitasnya juga $O(\log_3(n))$. Namun komplesitas binary heapify adalah $O(\log_2(n))$ & karena tinggi nya adalah $\log_2(n)$ sehingga karena $O(\log_3 n) = O(\log_2 n)$, maka komplesitas keduanya sama // (hanya berbeda pada konstanta basis log (nilai c selalu sama untuk nilai n apapun))

ⓒ
```
DELETE-ELEMENT(A, x):
    Pos = -1
    For i = 0 to len(A):  # Find the index of element x
        IF(A[i] == x):
            Pos = i
            break
    IF(Pos == -1):
        return
    A[Pos] = A[len(A)-1]  # change the value of A[Pos] basically deleting A[Pos] original
    len(A) -= 1
    IF A[Pos] < X
        TERNARY-MAX-HEAPIFY(A, Pos)
    else:
        TERNARY-MAX-PERCOLATE-UP(A, Pos)

TERNARY-MAX-PERCOLATE-UP(A, i):   # di sort keatas
    m = ⌈i/3⌉ - 1
    IF(m ≥ 0 and A[m] < A[i]):
        Swap(A[i], A[m])
        TERNARY-MAX-PERCOLATE-UP(A, m)
```

Pada proses heapify maupun percolate memiliki komplesitas sesuai dengan ketinggiannya yaitu $O(\log_3 n)$. Akan tetapi untuk melakukan proses pencarian elemen tersebut, kita butuh melakukan pencarian pada n element maka membutuhkan komplesitas sebesar $O(n)$.

Jadi:
- Apabila elemen belum diketahui posisinya membutuhkan $O(n)$
- Apabila sudah diketahui posisinya, membutuhkan $O(\log_3(n))$

Nama : Alvaro Austin
kelas : A
NPM : 2106729180
kode asisten : 3

d) COMBINE-HEAP(A1, A2):
result = concat(A1, A2) #
from i = len(result)-1 down to 0:
TERNARY-MAX-HEAPIFY(result, i) } lakukan heapify dari node terbawah
return result

Kompleksitas Combine Heap ini adalah O(N+M) dimana N adalah jumlah node A1 dan M adalah jumlah node A2
Pendapatan O(M+N) dapat dipisah menjadi 2 kasus karena contoh dilakukannya proses heapify sebanyak node adalah O(M). Pembuktiannya : (Anggap ada heap dengan jumlah node m ≠ 0 beda dengan M)

misalkan sebuah heap memiliki k level dimana level root adalah 0

Level	Jumlah node
0	1
1	3
2	9
⋮	
k-2	3^{k-2}
k-1	3^{k-1}

• sehingga banyak node dari 1 .. k adalah
$m = 1+3+9+\ldots+3^{k-1} = \frac{3^k - 1}{2}$

Dapat diobservasi juga bahwa jumlah pemanggilan fungsi heapify pada node level ke - i adalah k - i + 1
Sehingga jumlah pemanggilan fungsi heapify untuk semua node adalah

$S = (k-k)\,3^{k-1} + (k-(k-1))\,3^{k-2} + \ldots + (k-1)3^0$

level = k-1, jumlah = k ; level = k-2, maka jumlah = k-1

$S = 3^{k-1}\left(3^{-1} + 2\cdot 3^{-2} + 3\cdot 3^{-3} + \ldots + (k-1)\cdot 3^{-(k-1)}\right)$

dimana artinya $= 3^{k-1} \cdot \sum_{i=1}^{k-1} i\,(3)^{-i} < 3^{k-1} \sum_{i=0}^{\infty} i\,(\frac{1}{3})^i$

kita dapat memperoleh

$S = \frac{3^k}{4}$ dimana $m = \frac{3^k - 1}{2}$, sehingga $2m - 1 = 3^k$

$S = \frac{2m-1}{4}$

$S = O(m)$

Maka dari itu terbukti bahwa proses heapify semua node dan heap memiliki kompleksitas O(m) maka COMBINE-HEAP memiliki kompleksitas O(n+m)

Nama : Alvaro Agustin

NPM : 2106752180

Kelas : A

Kode Asisten : 3

a) Awal :

x = 1 A[x] = 3

i = 1

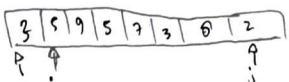
j = 9

Iterasi 1 :

kita lihat bahwa iterasi pertama pada iterasi 1 : → dan j ≤ p

a) repeat j = j - 1 sampai A[j] ≤ A[x] → kita dapat melihat karena 2 ≤ 3 maka j berubah (kali menjadi 8

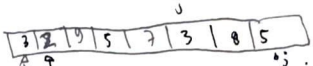
b) repeat i = i+1 sampai A[i] ≥ A[x] dan i ≥ r → akan stop ketika 5 ≥ 3 maka i sekarang pada posisi 2 :



i = 2

j = 8

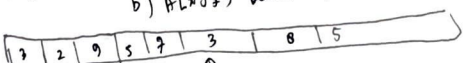
Swap :

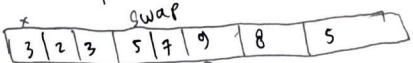


Iterasi 2 :

karena pada operasi a) A[x] = 3 lalu berhenti pada j = 6,

b) A[x] ≠ 3 berhenti pada i = 3.





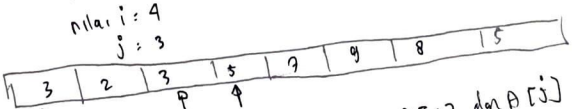
Iterasi 3 :

operasi a dijalankan sampai j bernilai 3 maka sekarang

b dijalankan sekali saja karena 5 ≥ 3 → A[x]

nilai i = 4

j = 3



loop selesai karena nilai i > j dan swap A[x] dan A[j]

swap(A[1], A[3])

(b) Tidak, dapat dilihat dari contoh diatas bahwa pada operasi terakhir elemen A[1] dan A[3] di tukar. hasil akhir : [3, 2, 3, 5, 7, 9, 8, 5]

Hal ini menyebabkan A[3] yg berasal dari A[6] (pada array original) terlebih sebelum A[1] (original) muncul terlebih dahulu dengan elemen yg muncul

Oleh karena itu, karena proses pertukaran elemen yg dibelakang menyebabkan algoritma ini tidak stable.

(c) RANDOMIZED-HOARE-PARTITION (A, p, r)

random_var : randomize (p, r) # get random partition index between p and r

swap (A[p], A[random_var]) # swap first element with pivot element to get randomized element that picked

return HOARE-PARTITION (A, p, r)

Perbandingan best case, worst case, dan juga average case pada implementasi ini tetaplah sama yaitu,

Best case : O(n log n)

Worst case : O(n²) → perbedaannya lebih randomized sehingga probabilitasnya sangatlah kecil.

Average : O(n log n)

→ Perbedaan randomized Hoare partition dengan Lomuto : Pada randomized hoare partition, biasanya secara rata-rata $\frac{1}{3}$, hoare partition lebih efisien karena melakukan swapping 3 kali lebih sedikit dibandingkan lomuto

→ Randomisasi pada partisi : untuk menurunkan kemungkinan terjadinya worst case scenario

Oleh karena itu, biasanya, average running time bagi randomized ini lebih di percaya lebih cepat