CSE-321 _ HW1

Umut Ay ALPER 1801042097

1. • $T_1(n) = 3\log n + 3$ → ignore constants $\equiv \log n$

• $T_2(n) = 4\log(\log n)$ → ignore constants $\equiv \log(\log n)$

L'hopital

$$\lim_{n\to\infty} \frac{\log n}{\log(\log n)} = \frac{1/\ln(n)}{1/\ln(n)\cdot\ln(n)} = \lim_{n\to\infty}(\ln(n)) = +\infty$$

↳ L'hopital $= +\infty$ means $f = \omega(g)$ = f grows faster than g

⇒ $T_2(n) = O(T_1(n))$ T_1 grows faster than T_2 or T_2 grows no faster than T_1

• $T_3(n) = n^5 + 8n^4 \rightarrow n^5$

• $T_4(n) = 2000n + 1 \rightarrow n$

$$\lim_{n\to\infty} \frac{n^5}{n} = \lim_{n\to\infty} n^4 = +\infty \Rightarrow T_4(n) = O(T_3(n))$$

T_4 grows no faster than T_3

• $T_5(n) = \left(\frac{n}{6}\right)^2 \rightarrow n^2$

• $T_6(n) = 3^n + n^2 \rightarrow \lim_{n\to\infty} \frac{3^n}{n^2} = +\infty$ (means 3^n grows faster than n^2) $= 3^n$

$$\lim_{n\to\infty} \frac{3^n}{n^2} = +\infty \quad \frac{T_5}{T_6}$$

$T_6 > T_5$

⇒ $T_5(n) = O(T_6(n))$

• $T_7(n) = n^n + 1000n \rightarrow n^n$

• $T_8(n) = 2^n + n^3 \rightarrow 2^n$

$$\lim_{n\to\infty} \frac{n^n}{2^n} = +\infty,$$ T_7 grows faster than T_8 (ω) T_8 grows no faster than T_7 (O)

⇒ $T_8(n) = O(T_7(n))$

• $T_2, T_3 \rightarrow \lim_{n\to\infty} \frac{\log(\log n)}{n^5} \Rightarrow T_2(n) = O(T_3(n))$

• $T_4, T_6 \rightarrow \lim_{n\to\infty} \frac{n}{3^n} \Rightarrow T_4(n) = O(T_6(n))$

• $T_5, T_7 \rightarrow \lim_{n\to\infty} \frac{n^2}{n^n} \Rightarrow T_5(n) = O(T_7(n))$

• $T_6, T_8 \rightarrow \lim_{n\to\infty} \frac{3^n}{2^n} \Rightarrow T_8(n) = O(T_6(n))$

• $T_3, T_5 \rightarrow \lim_{n\to\infty} \frac{n^5}{n^2} \Rightarrow T_5(n) = O(T_3(n))$

• $T_1, T_4 \rightarrow \lim_{n\to\infty} \frac{\log n}{n} \Rightarrow T_1(n) = O(T_4(n))$

• increasing order:

$$T_2 < T_1 < T_4 < T_5 < T_3 < T_8 < T_6 < T_7$$

2. a) $\lim_{n\to\infty} \frac{99n}{n} = \text{constant}$

$\underline{f \in \theta(g)}$

b) $\lim_{n\to\infty} \frac{2n^4 + n^2}{(\log n)^6} = \frac{8n^3}{\frac{6 \cdot \ln^5(n)}{n \cdot \ln 6}} = 4\ln(6)\, n^4$

$= 0$

(f grows slower than g) $\underline{f \in o(g)}$ little o

c) $\sum_{x=1}^{n} x = 1+2 \cdots x = \frac{n^2+n}{2} = \frac{n(n+1)}{2}$

$\Rightarrow \lim_{n\to\infty} \frac{n^2+n}{8n+2\log n} = \frac{n+1}{8 + \frac{2\ln(n)}{n}} = \frac{\infty}{\underset{(\text{constant})}{8}}$

$= \frac{\infty}{\text{constant}} = \frac{\infty}{c} = \infty \equiv \underline{f \in w(g)}$

f grows faster g

d) $\lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}} = \frac{3^n \ln(3)}{\frac{\ln(5) \cdot 5^{\sqrt{n}}}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{3^n}{5^n}$

rule: $\frac{a^c}{b^c} = \left(\frac{a}{b}\right)^c \to \lim_{n\to\infty} \left(\frac{3}{5}\right)^n = 0\ (a^x)$

$0 < a < 1 \Rightarrow 0 \Rightarrow \underline{f \in o(g)}$ little o

f grows slower than g

---

3. Program takes an integer array and length of it. as inputs. Then count element occurences in array, if any elements occurs more than half of length output will be this element, otherwise output is -1.

$\text{for}(i) \to \Omega_b(n),\ O_w(n^2)$

$\text{for}(j) \to \Omega_b(n),\ O_w(n) \Rightarrow \theta_{av}(n)$

$\left.\right\}$ $O_{best}(n)$, $O_{worst}(n^2)$ Time complexity

Space comlexity $= O(1)$

---

4. Program takes an integer array with its length as inputs. Then finds max element in array, After that it finds the frequency of each element that how many times ocurs in array. If any elements occurs more than half of length then output will be this number, else return -1..

$\text{for}(i_1) \to \theta_{av}(n)$

$\text{for}(i_2) \to \theta_{av}(n)$

$\text{for}(i_3) \to \Omega_a(n),\ O_w(n)$

Time complexity $= \theta_{av}(n)$

Space complexity $= O(n)$

---

5. Compare (3-4) : They have the same complexity for best cases, but in worst case seenerios Q3 has $O(n^2)$ worst case but Q4 has $O(n)$ worst case On the other hand Q3 has $O(1)$ constant space complexity but Q4 has $O(n)$ space complexity by using a map created.
Q4 uses and create extra memory by finding max element in array. that means that ~~thi~~ this will be very hard for memory if max number ~~would~~ will be very big, since allocation made according to this max num.

6. A = [a1,a2 --- an] , B = [b1,b2 --- bm]

a) fori(n) , find max{ai*bj} pseudo ⟹ code
forj(m)

```
foo (A,B)
    max ← A[0]*B[0]
    for i inrange n
    for j inrange m
        if (A[i]*B[j] > max)
            max ← A[i]*B[j]
    return max
```

Time complexity : for(i) and for(j) loops must be loop from 0-n, 0-m

$\Omega_b = n$

$O_w = n$ ⟹ $\Theta_{av}(n)$

b) //first concatenate two arrays

```
i,j,k ← 0
while (i<n) arr[k++] = A[i++]      → Θ(n)
while(j<m) arr[k++] = B[j++]       → Θ(m)
//then sort new concatenated array with size m+n

for(i to m+n)
    for(j to m+n)                  } Θ((m+n)²)
        if (ARR[j] < ARR[j+1])
            swap(ARR[j] and ARR[j+1])  → Θ(1)

exit//halt
```

$\Theta((m+n)^2)$ Time complexity

c) adding element → in best case adding must be O(1) but if array have to be duplicated size because of the insufficient size complexity will be O(n), but in worst case also duplication or inserting in size order has O(n) complexity, all the other scenerios has O(1) time.

d) deleting element → normally deleting an element from needs to be rearrange the array by filling the deleted space, since deleting is $O_w(n)$ but in best $\Omega(1)$ for last element deleted.

o

Umut Ay ALPER 1801042097