GEBZE TECHNICAL UNIVERSITY

INT RODUCTION TO ALGORITHM DESIGN

HW1

Merve Horuz

1801042651

Merve Horuz
18010426051

1) $\lim_{n\to\infty} \frac{T_1(n)}{T_2(n)} = \frac{3\log n + 3}{4\log n (\log n)} = \frac{3}{\frac{n\log 10}{4}} = \frac{3\log n}{4} = \infty$

$\frac{4}{n\log(10)\log(n)\log(10)}$

* $T_2(n) \in O(T_1(n))$

$\lim_{n\to\infty} \frac{T_1(n)}{T_3(n)} = \frac{3\log n + 3}{n^5 + 8n^4} = \frac{\frac{3}{n\log 10}}{5n^4 + 32n^3} = \frac{3}{n(5n^4+32n^3)} = 0$

* $T_1(n) \in O(T_3(n))$

$\lim_{n\to\infty} \frac{T_3(n)}{T_6(n)} = \frac{n^5+8n^4}{2222n+1} = \frac{5n^4+32n^3}{2222} = \infty$

* $T_4(n) \in O(T_3(n))$

$\lim_{n\to\infty} \frac{T_6(n)}{T_5(n)} = \frac{2222n+1}{\frac{n^2}{36}} = \frac{2222}{\frac{n}{18}} = \frac{38232}{n} = 0$

* $T_6(n) \in O(T_5(n))$

$\lim_{n\to\infty} \frac{T_3(n)}{T_8(n)} = \frac{n^5+8n^4}{2^n+n^3} = \frac{5n^4+32n^3}{\ln 2.2^n + 3n^2} = \frac{20n^3+96n^2}{(\ln 2)^2.2^n+6n} = \frac{60n^2+192n}{(\ln 2)^3.2^n+6} = \frac{120n+192}{(\ln 2)^4 2^n}$

$= \frac{120}{(\ln 2)^5 2^n} = 0$ * $T_3(n) \in T_8(n)$

$\lim_{n\to\infty} \frac{T_6(n)}{T_7(n)} = \frac{3^n+n^2}{n^n+1222n} = \frac{3^n\ln 3+2n}{n^n(\log n+1)+1222} = \frac{3^n.\ln 3.\ln 3+2}{n^n(\frac{1}{n}+(\log n+1)^2)} = 0$

* $T_6(n) \in T_7(n)$

$\lim_{n\to\infty} \frac{T_8(n)}{T_6(n)} = \frac{2^n+n^3}{3^n+n^2} = \frac{2^n.\ln 2 + 3n^2}{3^n.\ln 3 + 2n} = \frac{2^n.(\ln 2)^2 + 6n}{3^n(\ln 3)^2+2} = \frac{2^n.(\ln 2)^3+6}{3^n(\ln 3)^3} = \frac{2^n.(\ln 2)^4}{3^n.(\ln 3)^4}$

$= 0 \Rightarrow$ * $T_8(n) \in O(T_6(n))$

* $T_2(n) < T_1(n) < T_4(n) < T_5(n) < T_3(n) < T_8(n) < T_6(n) < T_7(n)$

2) if limit solution is 0, then $f(n) \in O(g(n))$
if " " " ∞, then $f(n) \in \Omega(g(n))$
if " " " constant, " $f(n) \in \Theta(g(n))$

a) $\lim_{n\to\infty} \frac{99n}{n} = 99 \Rightarrow f(n) \in \Theta(g(n))$

b) $\lim_{n\to\infty} \frac{2n^4+n^2}{(\log n)^6} = \frac{8n^3+2n}{\frac{6(\log^5(n)}{n}} = \frac{8n^4+2n^2}{6\log^5(n)} = \frac{32n^3+4n}{30\log^4(n)} \ldots\ldots = \infty$

$\Rightarrow f(n) \in \Omega(g(n))$

c) $\sum_{x=1}^{n} x = 1+2+3..n = \frac{n.(n+1)}{2} \Rightarrow \lim_{n\to\infty} \frac{\frac{n^2+n}{2}}{4n+\log n} = \frac{2n+1}{8+\frac{2}{n}} = \frac{(2n+1)n}{8n+2} = \frac{4n+1}{8}$

$= \infty \Rightarrow f(n) \in \Omega(g(n))$

d) $\lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}}$ = On the back page!

$\lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}} \Rightarrow \frac{3^n \ln 3}{\frac{d5^u}{du}\frac{du}{dx}}$ → where $u = \sqrt{x}$ and $\frac{d}{du}(5^u) = 5^u \ln 5$:

$= 5^{\sqrt{x}} \ln(5) \left(\frac{d}{dx}(\sqrt{x})\right)$

⋮

and we know that $a^n \subset a^{n^b}$ for $b > 1$ so $a^{n^b} \subset a^n$

So $\lim_{n\to\infty} \frac{3^n}{5^{\sqrt{n}}} = \infty$

then $f(n) \in \Omega(g(n))$

3) a) There is an array (of size N) with an element repeated more than N/2 number of time and the rest of the element in the array can also be repeated but only one element is repeated more than N/2 times. And this element is called majority element, this algorithm finds majority element.
For example: nums [ ] => [1,2,3,6,5,5,5,5,5,5]
then when i=4 and start from 5 it set count variable to 6.
and 6 > 10/2, so, returns 5 (majority element).

b) There is two main work, (for loops)
if there is not majority element then the worst case scenario is occured. for example; nums = [1,2,3,6,5] in this array it looks 4 element and 3 element .....
4+3+2+1 → so $\frac{(n-1).n}{2}$ => $\frac{n^2-n}{2}$ eliminate lower elements.
=> n^2 => $O(n^2)$ → worst case

best case scenario, : it finds directly majority element. for example: nums = [3,3,3,3,3]
so it looks 4 elements (for loop only works four times)
=> n-1 times. => $\Omega(n)$ → best case

4) a) This algorithm finds majority element too. That is same work in the question 3. There is an array (of size N with an element repeated more than N/2 number of time and the rest of element in the array can also be repeated but only one element is repeated more than N/2 times.
for example nums array is {1,1,1,1,2,3} ← input.
max = 3 map = {0,4,1,1} so 4 > 6/2 => returns 1.
output = 1.

b) first and second for loop iterates n times.
third for loop iterates n or less than n times.
So best case and worst case are O(n) time complexity.

5) In wast case, algorithm in question 3 $O(n^2)$ but algorithm in question 4 $O(n)$ so algorithm in question 4 is better than algorithm in question 3.
Algorithm in question 3 fixed space is used. $O(1)$
Algorithm in question 4 sometimes takes a lot of space, sometimes takes a little of memory space. For example nums[] = {1,2,11020} allocate 11021 memory space but algorithm in question 3 allocate only 3 memory space. On the other hand, nums [] = {1,1,1,1, ... 1} algorithm in question 4 allocate only 2 memory space but algorithm in question 3 allocate how many 1s are there, in this case " in question 4 is better...

* My Time complexity makes better than algorithm in question 4.
* space complexity makes " " in question 3.
* space complexity of algorithm in question 3 is $O(1)$.
* space complexity of algorithm in question 4 is $O(n)$.

6)

a)
```
int findMax (int nums[], int n){
    int max=0, i;
    for (i=0; i<n; i++)
        if (nums[i] > max)
            max = nums[i];

    return max;
}
max = findMax(A,n) * findMax(B,m);
```
finds max of two list and multiplies them.

b) find max in the first array and keep index of this element. (1)
find max in the second array and keep index of this element. (2)
compare max in the first array and max in the second array (3)
if max1 > max2, store max1 in the last array. (4)
change element index of max1 to '*' (dummy char). (5)
if max2 > max1, store max2 in the last array (6)
change element index of max1 to '*' (dumy char). (7)
iterate this algorithm (m+n) times (length of two arrays). (8)

c)
use dynamic array for this algorithm.
reallocate array with n+1 size where n is the size of array.
store new element nth index of array.

d) allocate new array with n-1 size. (1)
Store all elements except that will removed element. (2)
determine new array as current array. (3)

6) Time complexity of algorithms.

a) Time complexity of findMax function $O(n)$.
There are two arrays so length one of them is n, length one of them is m. So total complexity is $O(n+m)$ if $m>n$ final time complexity $O(m)$, if $n>m$ final time complexity $O(n)$.

b) first matter takes n times (1)
second matter takes m times due to for loop (2)
eighth matter takes n+m times. (8)
so total time complexity $O((n+m).(n+m))$.
$\Rightarrow O(n^2 + 2nm + m^2)$
$\Rightarrow$ if n^2 > all other elements
$\Rightarrow O(n^2)$ //
else if m^2 > all other elements
$O(m^2)$ //
else if $2nm$ > all other elements
$O(2nm)$ //

n → size of first array
m → size of second array

c) In good cases realloc returns same pointer (so $O(1)$) and in less happy cases it need to copy the memory zone elsewhere.
We consider worst case $O(n)$.

d) In most heap implementations, n is the number of contiguous chunks of memory the manager is handling. This is decidedly not something typically under client control. So really, unless you are implementing a heap, then the proper answer is that it is non-deterministic.
for second matter in pseudocode, takes n time due to storing all elements in other array.
So, total time complexity for deleting an item from array
$O(n)$.

Algorithm for 6b:

```
void descendingOrder(int last[], int first[], int second[], int n, int m, int *s){

    int i, max1=0, max2=0, k, j, l;

    for(i=0;i<n;i++)
        if(first[i] != '*' && first[i] > max1){
            max1 = first[i];
            j = i;-
        }

    for(k=0;k<m;k++)
        if(second[k] != '*' && second[k] > max2){
            max2 = second[k];
            l = k;
        }

    if(max1 > max2){
        last[(*s)++] = first[j];
        first[j] = '*';
    }
    else{
        last[(*s)++] = second[l];
        second[l] = '*';
    }
}
```

Algorithm for 6d:

```
void removeElement(int arr[], int a, int n){

    int * arr2 = (int*) malloc((n-1) *sizeof(int));

    for(int i=0, k=0;i<n;i++)
        if(arr[i] != a){
            arr2[k] = arr[i];
            k++;
        }

    arr = arr2;
    for(int i=0;i<5;i++)
        printf("%d",arr[i]);
}
```