Gebze Technical University
Department of Computer Engineering
CSE 321 Introduction to Algorithm Design
Fall 2020
Midterm Exam (Take-Home)
November 25th 2020-November 29th 2020

Student ID and Name	Q1 (20)	Q2 (20)	Q3 (20)	Q4 (20)	Q5 (20)	Total
171044098 Akif Kartal						

Read the instructions below carefully

- You need to submit your exam paper to Moodle by November 29th, 2020 at 23:55 pm as a single PDF file.
- You can submit your paper in any form you like. You may opt to use separate papers for your solutions. If this is the case, then you need to merge the exam paper I submitted and your solutions to a single PDF file such that the exam paper I have given appears first. Your Python codes should be in a separate file. Submit everything as a single zip file.

Q1. List the following functions according to their order of growth from the lowest to the highest. Prove the accuracy of your ordering. **(20 points)**

Note: Your analysis must be rigorous and precise. Merely stating the ordering without providing any mathematical analysis will not be graded!

a) 5^n

b) $\sqrt[4]{n}$

c) $\ln^3(n)$

d) $(n^2)!$

e) $(n!)^n$

Q2. Consider an array consisting of integers from 0 to n; however, one integer is absent. Binary representation is used for the array elements; that is, one operation is insufficient to access a particular integer and merely a particular bit of a particular array element can be accessed at any given time and this access can be done in constant time. Propose a linear time algorithm that finds the absent element of the array in this setting. Rigorously show your pseudocode and analysis together with explanations. Do not use actual code in your pseudocode but present your actual code as a separate Python program. **(20 points)**

Q3. Propose a sorting algorithm based on quicksort but this time improve its efficiency by using insertion sort where appropriate. Express your algorithm using pseudocode and analyze its expected running time. In addition, implement your algorithm using Python. **(20 points)**

Q4. Solve the following recurrence relations

a) $x_n = 7x_{n-1} - 10x_{n-2}$, $x_0=2$, $x_1=3$ **(4 points)**

b) $x_n = 2x_{n-1} + x_{n-2} - 2x_{n-3}$, $x_0=2$, $x_1=1$, $x_2=4$ **(4 points)**

c) $x_n = x_{n-1} + 2^n$, $x_0=5$ **(4 points)**

d) Suppose that a^n and b^n are both solutions to a recurrence relation of the form $x_n=\alpha x_{n-1}+\beta x_{n-2}$. Prove that for any constants c and d, ca^n+db^n is also a solution to the same recurrence relation. **(8 points)**

Q5. A group of people and a group of jobs is given as input. Any person can be assigned any job and a certain cost value is associated with this assignment, for instance depending on the duration of time that the pertinent person finishes the pertinent job. This cost hinges upon the person-job assignment. Propose a polynomial-time algorithm that assigns exactly one person to each job such that the maximum cost among the assignments (not the total cost!) is minimized. Describe your algorithm using pseudocode and implement it using Python. Analyze the best case, worst case, and average-case performance of the running time of your algorithm. **(20 points)**

CSE 321 Fall 2020 Middterm Exam Solutions

171044098

Akif Kartal

Q1) Firstly, we need to order given functions. If we order them we will get;

$$\boxed{\begin{array}{c} \ln^3(n) < \sqrt[4]{n} < 5^n < (n!)^n \leq (n^2)! \\ c < b < a < e \leq d \end{array}}$$

Now we need to prove 4 equality.

1) $c \in O(b) \Rightarrow \ln^3(n) \in O(\sqrt[4]{n})$

By using the method of taking limit and L'hospital rule;

$$\lim_{n\to\infty} \frac{\ln^3(n)}{n^{1/4}} = \lim_{n\to\infty} \frac{(\ln^3(n))'}{(n^{1/4})'} = \lim_{n\to\infty} \frac{\frac{3\ln^2 n}{n}}{\frac{1}{4}n^{-3/4}}$$

$$= \lim_{n\to\infty} \frac{(12\ln^2 n)'}{(n^{7/4})'} = \lim_{n\to\infty} \frac{\frac{24\ln(n)}{n}}{\frac{7n^{3/4}}{4}} = \lim_{n\to\infty} \frac{96\ln(n)}{7n^{7/4}}$$

$$= \lim_{n\to\infty} \frac{(96\ln(n))'}{(7n^{7/4})'} = \lim_{n\to\infty} \frac{\frac{96}{n}}{\frac{49n^{3/4}}{4}} = \lim_{n\to\infty} \frac{384}{49n^{7/4}} = 0$$

Thus, if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ means that $f(n) \in O(g(n))$.

2) $b \in O(a) \Rightarrow \sqrt[4]{n} \in O(5^n)$

By using the method of taking limit and L'hospital Rule;

$$\lim_{n\to\infty} \frac{n^{1/4}}{5^n} = \lim \frac{(n^{1/4})'}{(5^n)'} = \lim_{n\to\infty} \frac{\frac{1}{4n^{3/4}}}{\ln(5).5^n}$$

$$= \lim_{n\to\infty} \frac{1}{4n^{3/4}.\ln(5).5^n} = 0$$

---

3) $a \in O(e) \Rightarrow 5^n \in O((n!)^n)$

By using the method of taking limit

$$\lim_{n\to\infty} \frac{5^n}{(n!)^n} = \left(\begin{array}{c}\text{By using stirling's}\\ \text{approximation}\end{array}\right) = \lim_{n\to\infty} \frac{5^n}{\left(\sqrt{2\pi n}\cdot\left(\frac{n}{e}\right)^n\right)^n}$$

$$\lim_{n\to\infty} \frac{5^n . e^{n^2}}{(\sqrt{2\pi n})^n . n^{n^2}} = \lim_{n\to\infty} \left(\frac{5}{\sqrt{2\pi n}}\right)^n \cdot \left(\frac{e}{n}\right)^{n^2} = 0$$

---

4) $d \in O(e) \Rightarrow (n^2)! \in O(n!)^n$

By using the method of taking limit

$$\lim_{n\to\infty} \frac{(n^2)!}{(n!)^n} =$$ By using stirling's approximation

Scanned with CamScanner

Stirling approximation:

$$n! \cong \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\lim_{n\to\infty} \frac{(n^2)!}{(n!)^n} = \lim_{n\to\infty} \frac{\left(\sqrt{2\pi n}\cdot\left(\frac{n^n}{e^n}\right)\right)^n}{\sqrt{2\pi n^2}\cdot\left(\frac{n^2}{e}\right)^{n^2}}$$

$$= \lim_{n\to\infty} \frac{\sqrt{2\pi n}}{n\cdot\sqrt{2\pi}\cdot(n)^{n^2}} = \lim_{n\to\infty} \frac{2\pi \cancel{n}}{\cancel{n}^2\cdot 2\pi\cdot\left(n^{n^2}\right)^2}$$

$$= \lim_{n\to\infty} \frac{2\pi}{n\cdot 2\pi\cdot\left(n^{n^2}\right)^2}$$ = So when it goes to infinity n will be bigger than constant 2π then result will be 0.

Thus, $\lim_{n\to\infty} \frac{(n^2)!}{(n!)^n} = 0$

Q2) In order to solve this question, since numbers is in binary representation we can use binary number properties. As a binary number property;

⇒ If the last digit of a binary number is 1, the number is odd, if it is 0, the number is even.

For example;

1011 represent an odd number 11
0110 represent an even number 6

Thus, by using Last bit of binary number we can easily achieve our goal in linear time.

My algorithm steps

1) check place of number in array (which index)
   - if place is even index
     * Last digit of that binary number must be 0.
   - if place is odd index
     * Last digit of that binary number must be 1.
2) Repeat this n times if any rule violated then return that index as absent element.

Assumption: When I write this algorithm I assume each element of array is a string which is binary representation of that number in n-bit system. So, in this way I can easily access last element of that string since it is a char array in O(1) constant time.

Scanned with CamScanner

Pseudocode

```
procedure findabsent(A, n)
    for i = 0 to n do                              → O(n)
        size = get size of A[i]                    → O(1)
        lastbit = get A[i][size-1] element         → O(1)
        if i is even
            if lastbit is 1
                return i
            end if
        else // i is odd
            if lastbit is 0                        → O(1)
                return i
            end if
        end if
    end for
    return -1 // if no absent
end
```

$$+ \quad \overline{O(n)}$$

Analysis of algorithm

As you see from pseudocode we have only one loop and other statements takes $O(1)$ constant time.

Note that getting size of an array will take $O(1)$ time. Instead of this I could use a constant number for bit number but we don't need this. Thus total we have.

$$\boxed{T(n) = \Theta(n)}$$

(Programming languages keep a space to store size of array.)

Q3) We know that quicksort worst-case performace is $\Theta(n^2)$ which is quadratic. We have this bad performance because of 2 reason.

1) if array is almost sorted
2) if the number of elements is very small.

For 1st reason we need to better partition algorithm such as median partition but for 2nd reason we will use insertion sort when array size is small. In this way our performance will be improved.

Pseudocode

```
procedure quicksort_improved (A, l, h)
    if l < h
        if (h-l) < 25
            insertionSort (A, l, h+1)
        else
            p = Partition (A, l, h)
            quicksort_improved(A, l, p-1)
            quicksort_improved(A, p+1, h)
        end if
    end if
end
```

In this pseudocode we are checking array size each function call so that if array size is less than 25 we will use insertion sort instead of quick sort. (25 is randomly choosen constant size which is start date of this exam.)

```
procedure insertionsort(A, l, h)
    for i = l+1 to h do
        current = A[i]
        pos = i-1
        while (pos ≥ 0) and (current < A[pos]) do
            A[pos+1] = A[pos]
            pos = pos-1
        end while
        A[pos+1] = current
    end for
end
```

```
procedure partition(A, l, h)
    pivot = A[l]
    i = l
    j = h
    while (i<j)
        while (A[i] <= pivot)
            i = i+1
        end while
        while (A[j] > pivot)
            j = j-1
        end while
        if i<j
            swap A[i] with A[j]
        end if
    end while
    swap A[l] with A[j]
end    return j
```

Analyzing running time

If we analyze this new algorithm with a small change in time functions terms there is no difference. Quick sort has still worst case $\Theta(n^2)$, since insertion sort average case $\Theta(n^2)$. But this is in function term, if you do a real life experiment, with this new algorithm, you will see that running time improved with this new version of quick sort algorithm.

Thus, we have;

$T_{best}(n) = \Theta(n\log n)$ $T_{worst}(n) = \Theta(n^2)$ $T_{avg}(n) = \Theta(n\log n)$	But, even if our complexity doesn't change, we will have efficient algorithm because we will have less number of comparison and less space complexity for small arrays.

Analyzing space complexity

Since Quick sort is a divide and conquer algorithm for better performance it consumes more space than insertion sort. Thus, since we used insertion sort our space complexity will decrease by this new version of quicksort algorithm.

Q4)

a) In order to solve this relation, I will use "Characteristic root technique." So this is a homogeneous recurrence relation therefore;

$$x_n = x_n^{(h)}$$

$$x_n = 7x_{n-1} - 10x_{n-2}, \quad x_0 = 2, \; x_1 = 3$$

Characteristic equation $= r^2 = 7r - 10r$

$$r^2 - 7r + 10 = 0$$

$$(r-5)(r-2) = 0$$

$\boxed{\begin{matrix} r_1 = 5 \\ r_2 = 2 \end{matrix}}$ (roots

Since $r_1 \neq r_2$ our equation is

$$\boxed{x_n^{(h)} = c_1(5)^n + c_2(2)^n}$$

$x_0 = 2, \; x_1 = 3$

$-2 / \; 2 = c_1 + c_2$

$+ \; 3 = 5c_1 + 2c_2$

$-4 = -2c_1 - \cancel{2c_2}$

$+ \; 3 = 5c_1 + \cancel{2c}$

$-1 = 3c_1$

$\boxed{c_1 = -\frac{1}{3}}$

$c_1 + c_2 = 2$

$-\frac{1}{3} + c_2 = 2$

$\boxed{c_2 = \frac{7}{3}}$

$\rightarrow x_n^{(h)} = c_1(5)^n + c_2(2)^n$

$x_n^{(h)} = -\frac{1}{3}(5)^n + \frac{7}{3}(2)^n$

$$\boxed{x_n = -\frac{1}{3}(5)^n + \frac{7}{3}(2)^n}$$

(result)

b) To solve this question, I will use characteristic root technique. This recurrence relation is homogenous relation Thus;

$$x_n = 2x_{n-1} + x_{n-2} - 2x_{n-3}, \quad x_0 = 2, x_1 = 1, x_2 = 4$$

Characteristic equation $= r^3 = 2r^2 + r - 2$

$$\Rightarrow r^3 - 2r^2 - r + 2 = 0$$

$$\Rightarrow r^2(r-2) - (r-2) = 0$$

$$\Rightarrow (r-2).(r^2-1) = 0$$

$$\Rightarrow (r-2)(r-1)(r+1) = 0$$

$$\Rightarrow r_1 = 2, r_2 = 1, r_3 = -1$$

$$x_n = c_1 . 2^n + c_2 . (1)^n + c_3 . (-1)^n$$

$$\boxed{x_n = c_1 . 2^n + c_2 + c_3 (-1)^n}$$

$x_0 = 2, \quad x_1 = 1, \quad x_2 = 4$

$2 = c_1 + c_2 + c_3 \rightarrow \boxed{c_2 = 2 - c_1 - c_3}$

$1 = 2c_1 + c_2 - c_3$ subsitute here

$4 = 4c_1 + c_2 + c_3$

$2c_1 + 2 - c_1 - c_3 - c_3 = 1$

$\boxed{c_1 - 2c_3 = -1}$

$4c_1 + 2 - c_1 - \cancel{c_3} + \cancel{c_3} = 4 \Rightarrow 3c_1 = 2$

$\boxed{c_1 = \frac{2}{3}}$

$c_1 - 2c_3 = -1$

$\frac{2}{3} - 2c_3 = -1$

$2c_3 = \frac{5}{3}$

$\boxed{c_3 = \frac{5}{6}}$

$c_2 = 2 - c_1 - c_3$

$c_2 = 2 - \frac{2}{3} - \frac{5}{6}$

$\boxed{c_2 = 1/2}$

$\Rightarrow$ To sum up

$$x_n = c_1 \cdot 2^n + c_2 + c_3 (-1)^n$$

$$c_1 = \frac{2}{3}, \quad c_2 = \frac{1}{2}, \quad c_3 = \frac{5}{6}$$

$$x_n = \frac{2}{3} \cdot 2^n + \frac{1}{2} + \frac{5}{6} (-1)^n$$

$$\boxed{x_n = \frac{2^{n+2} + 3 + 5 \cdot (-1)^n}{6}}$$

c) This relation is not homogeneous relation;

Thus;

$$x_n = x_n^{(h)} + x_n^{(p)} \quad (h = \text{homogeous},\ p = \text{private})$$

Let's find first homogenous solution

$$x_n = x_{n-1} + 2^n, \quad x_0 = 5$$

Characteristic equation $= r = 1$

$$\boxed{x_n^{(h)} = c_1 1^n}$$

Let's find private solution

$$x_n^{(p)} = A.2^n \quad = 2.2^n$$

$$\boxed{x_n^{(p)} = 2^{n+1}}$$

$$x_{n-1} = A.2^{n-1}$$

$$\boxed{x_n = x_{n-1} + 2^n}$$

$$\Rightarrow A.2^n = A.2^{n-1} + 2^n$$

$$\Rightarrow A.2^n - \frac{A.2^n}{2} = 2^n$$

$$\Rightarrow \frac{A.2^n}{2} = 2^n$$

$$\Rightarrow A.\cancel{2^n} = \cancel{2^n}.2 \Rightarrow \boxed{A = 2}$$

$$x_n = x_n^{(h)} + x_n^{(p)}$$

$$x_n = c_1.1^n + 2^{n+1}$$

$$x_0 = 5$$

$$5 = c_1 + 2$$

$$c_1 = 3$$

$$x_n = 3.1^n + 2^n$$

$$\boxed{x_n = 2^n + 3} \text{ (result)}$$

d) Since a^n and b^n are both solution this relation, we can easily change them with X_n in recurrence relation.

$$a^n = a^n$$
$$a^{n-1} = a^{n-1}$$
$$a^{n-2} = a^{n-2}$$ change them with X_n

$$X_n = \alpha X_{n-1} + \beta X_{n-2}$$

$$\boxed{a^n = \alpha \cdot a^{n-1} + \beta a^{n-2}}$$

$$b^n = b^n$$
$$b^{n-1} = b^{n-1}$$
$$b^{n-2} = b^{n-2}$$

$$X_n = \alpha X_{n-1} + \beta X_{n-2}$$

$$\boxed{b^n = \alpha b^{n-1} + \beta b^{n-2}}$$

After that, to prove $ca^n + db^n$ is a solution of this recurrence relation, we have to multiply these relations with consants and then, we will sum them;

$$c \cdot a^n = c \cdot \alpha a^{n-1} + c \cdot \beta a^{n-2}$$
$$+ \; d \cdot b^n = d \alpha b^{n-1} + d \cdot \beta b^{n-2}$$

$$\boxed{c \cdot a^n + d \cdot b^n = \alpha \left(c \cdot a^{n-1} + d b^{n-1}\right) + \beta \left(c \cdot a^{n-2} + d \cdot b^{n-2}\right)}$$

Thus, since a^n and b^n is a solution of this relation, $c \cdot a^n + d \cdot b^n$ is also a solution of this recurrence relation.

Q5) In this question since we have 2 array (people and Jobs) and also we have costs hinges upon this person-Job assignment for this costs we need to costs Matrix as an input with arrays.

For instance;

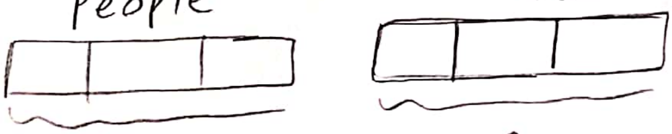
Job \ People	0	1	2
0	10	5	12
1	40	80	1
2	30	15	22

⇒ Costs for person-Job assignment

$n \times n$ ⇒ n by n matrix

Also we have;

People Job



(n is person and Job size)

n n

Question says minimized maximum cost while assinging Job to people.

To solve this question, my Algorithm is;

1) Check costs for each Job, if all maximum costs has same person then Just assing the Jobs in any order in this case number of maximum cost is will be 1.

2) if maximum costs are distrubuted on people then assing Jobs to people such that number of maximum cost will be 0 (zero).

Pseudocode

```
Procedure Job-assingment(People, Jobs, cost_matrix, n)
      // First find maximum cost indexes for each row in matrix.
O(1) <-  max-cost-indexes[]
         max-index = -1
         all-max-has-same-person = true
O(n^2) <- for i = 0 to n do
             max = -1
             for j = 0 to n do
                if max < cost[i][j]
                     max = cost[i][j]
                     max-index = j
                end if
             end for
             max_cost_indexes[i] = max-index
             if i != 0 and max-cost-indexes[i-1] != max_index
                    all-max-has-same-person = false
             end if
          end for
O(n) <-  if all-max-has-same-person
            // number of max cost is 1
            // assing jobs in any orde
            for i = 0 to n do
                 Jobs[i] = i
                 people[i] = i
            end for
         else
O(n^3) <-      assing-Jobs(People, Jobs, max-cost-indexes, n)
         end if
end
```

pseudocode cont'd

```
procedure assing-Jobs(People, Jobs, max-cost-indexes, n)
    for i = 0 to n do                                              O(n^3) <-
        if i == n-1 // if last index
O(n^2) <-   Jobs[i] = find-person(Job, n, max-cost-indexes[i])
        else
            is-found = false
            for k = i to n do                                      O(n^3) <- O(n^2) <-
                if (max-cost-indexes[k] != max-cost-indexes[i])    O(n) <-
    O(1) <-         if not is-found
    O(n) <-             if not Jobs.contain(max-cost-indexes[k])
                            Jobs[i] = max-cost-indexes[k]           O(1) <-
                            is-found = true
                        end if
                    end if
                end if
            end for
            if not is-found
O(n^2) <-       Jobs[i] = find-person(Job, n, max-cost-indexes[i])  O(n^3) <-
            end if
        end if
O(1) <- People[Jobs[i]] = i
    end for
end
```

⇒ Note that number of maximum cost is O(zero) in this case of algorithm.

Total time = $T(n) = \Theta(n^3)$ (for this function)

pseudocode cont'd

```
procedure find-person(Jobs, n, current-index)

O(1) <- person = -1
        for i=0 to n do
    O(n) <- if (not Jobs.contain(i)) and (i != current-index)
                        person = i
              end if
           end for
O(1) <- return person
end
```

$O(n^2) \leftarrow$ (for loop)

Note = contain method works $O(n)$ time since it makes search on array.

Total time $\Rightarrow T(n) = \Theta(n^2)$ (for this function)

General Analysis

Best Case

If same person has all maximum costs we will have $\boxed{T(n) = \Theta(n^2)}$ (to find maximum cost indexes in matrix. # of max cost is 1.)

Worst Case

If maximum costs are distributed on people we will have $\boxed{T(n) = \Theta(n^3)}$ time to assing Jobs on people with minimited maximum cost such that # of maximum cost is O(zero).

Average case

In average case, since the possibility of same person has all maximum cost is very low, we will have $\boxed{T(n) = \Theta(n^3)}$ running time.