yassin Algorithm section 1

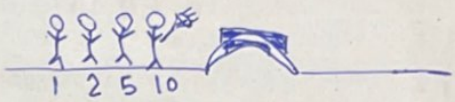
Section 1

Ex1.1

17 min

not now 4.Design an algorithm for computing √n for any positive integer n. Besides assignment and comparison, your algorithm may only use the four basic arithmetical operations.

6a. Find gcd(31415, 14142) by applying Euclid's algorithm.



Ex1.2

2.New World puzzle There are four people who want to cross a rickety bridge; they all begin on the same side. You have 17 minutes to get them all across to the other side. It is night, and they have one flashlight. A maximum of two people can cross the bridge at one time. Any party that crosses, either one or two people, must have the flashlight with them. The flashlight must be walked back and forth; it cannot be thrown, for example. Person 1 takes 1 minute to cross the bridge, person 2 takes 2 minutes, person 3 takes 5 minutes, and person 4 takes 10 minutes. A pair must walk together at the rate of the slower person's pace. (Note: According to a rumor on the Internet, interviewers at a well-known software company located near Seattle have given this problem to interviewees.)

9. Consider the following algorithm for finding the distance between the two closest elements in an array of numbers. ALGORITHM MinDistance(A[0..n − 1])

//Input: Array A[0..n − 1] of numbers

//Output: Minimum distance between two of its elements

dmin ← ∞

for i ← 0 to n − 1 do

for j ← 0 to n − 1 do if i = j and |A[i] − A[j]| < dmin

dmin ← |A[i] − A[j]|

return dmin

for i ← 0 to n-2 do
for j ← i+1 to n-1 do
if |A[i] - A[j]| < dmin
dmin ← |A[i] - A[j]| < dmin
return dmin

Make as many improvements as you can in this algorithmic solution to the problem. If you need to, you may change the algorithm altogether; if not, improve the implementation given

Ex1.3

1. Consider the algorithm for the sorting problem that sorts an array by counting, for each of its elements, the number of smaller elements and then uses this information to put the element in its appropriate position in the sorted array:
   ALGORITHM ComparisonCountingSort(A[0..n − 1]) //Sorts an array by comparison counting
   //Input: Array A[0..n − 1] of orderable values
   //Output: Array S[0..n − 1] of A's elements sorted
   // in nondecreasing order

A:

21	9	8	10

Count:

0	0	0	0

i j

S:


```
for i ← 0 to n − 1 do Count[i]← 0
for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
                if A[i] < A[j ] Count[j ]← Count[j ] + 1
                else Count[i]← Count[i] + 1
for i ← 0 to n − 1 do S[Count[i]]← A[i]
return S
```

a. Apply this algorithm to sorting the list 60, 35, 81, 98, 14, 47.

b. Is this algorithm stable? not now

c. Is it in-place?

Ex.1.4

1. Describe how one can implement each of the following operations on an array so that the time it takes does not depend on the array's size n.

a. Delete the ith element of an array ($1 \le i \le n$).

3	5	1	2

then decrease size of Array by one

3. a. Show the stack after each operation of the following sequence that starts with the empty stack: push(a), push(b), pop, push(c), push(d), pop

b. Show the queue after each operation of the following sequence that starts with the empty queue: enqueue(a), enqueue(b), dequeue, enqueue(c), enqueue(d), dequeue

d
c
b
a

c
a

ch.2 important

(3) b

abcd

section 2

Sheet 1

Q1: what is computing algorithms?

Q2: compute the following sums

a. 1+2+3+.....+70 → $\frac{(1+70)\,70}{2}$ = total sum

b. 1+3+5+.....+999 → $\frac{(1+999)\,500}{2}$ = total sum

c. $\sum_{i=3}^{5} 2$

d. $\sum_{i=3}^{10} C$

e. $\sum_{i=3}^{n} 2$

f. $\sum_{i=0}^{n-1} i$

g. $\sum_{i=0}^{n-1} n$

h. $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1$

i. $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} j$

j. $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} i$

k. $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k}^{n-1} k$

l. $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} i \cdot j$ ***

m. $\sum_{i=0}^{n-1} \sum_{j=0}^{i} 1$

n. $\sum_{i=0}^{n-1} \sum_{j=0}^{i} j$ ***

Problem → Algorithm → INPUT → Compiler → OUTPUT

$\sum_{i=K}^{n} C = (n-K+1)C$

$f = 0$, $L = n$, $S = n-0+1$, $n+1$

$\sum_{i=0}^{n} i = \frac{(0+n)\,n+1}{2}$

Q3: in the following code segments define the basic operation and use mathematical analysis to prove how many times the basic operation was executed: T(N)

a.
```
x<-0
for i<- 1 to n
        x<- x+1
end
```
$\sum_{i=1}^{n} 1 = n1$, $T(n) = n$

b.
```
x<-0
for i<- 1 to n
        x<- x+1*2
end
```
$T(n) = \sum_{i=1}^{n} 1 = n$

c.
```
x<-0
for i<- 1 to n
        x<- x+1
        x<- x+x
end
```
$T(n) = \sum_{i=1}^{n} 2 = 2n$

d.
```
x<-19
for i<- 1 to n
        x<- x-1
        x<- x+x
end
```
$T(n) = \sum_{i=1}^{n} 2 = 2n$

e.
```
x<-0
for i<- 1 to n
        for j<-1 to n
```
$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} 2 = n * n$, $2n^2$

*, /

+, −

>=, <=, <, >, !=, ==

:=

Scanned with CamScanner

```
            x<- x+1
            x<- x+x
        end

    end
```

f.
```
x <-0
for i <- 1 to n
        for j<-1 to i
            x<- x*2
            x<- x/3
        end

end
```

$$\sum_{i=1}^{n} \sum_{j=1}^{i} 2 = \sum_{i=1}^{n} 2i$$

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

$f = 1$

$l = 1$

$s = n$

$$\frac{2(n+1)n}{2}$$

$$n^2+n$$

Rules:

a. $\sum_{i=k}^{n} C$ = (n-k+1) C

b. $1 + 2 \ldots\ldots\ldots + n = \frac{(first+last)size}{2} = \frac{(1+n)n}{2}$

section 3

Sheet 2 (Ch2: 2.1→2.2)

Q1: Consider the definition-based algorithm for adding two n × n matrices.

a. What is its basic operation? How many times is it performed as a function of the matrix order n? As a function of the total number of elements in the input matrices?

b. Answer the same questions for the definition-based algorithm for matrix multiplication.

Q2: Glove selection There are 22 gloves in a drawer: 5 pairs of red gloves, 4 pairs of yellow, and 2 pairs of green. You select the gloves in the dark and can check them only after a selection has been made. What is the smallest number of gloves you need to select to have at least one matching pair in the best case? In the worst case?

1R 1L (2) 11R 1L (12)

Q3: For each of the following functions, indicate how much the function's value will change if its argument is increased fourfold.

a. $\log_2 n$ b. $\sqrt{n}$ c. n d. n^2 e. n^3 f. 2^n

$\log_2 n \to \log_2 4n = \log_2 4n - \log_2 n$

$= \log_2 4 + \log_2 n - \log_2 n = 2$

$\frac{\sqrt{4n}}{\sqrt{n}} = \frac{2\sqrt{n}}{\sqrt{n}} = 2$

$\frac{4n}{n} = 4$

$\frac{4n^2}{n^2} = 16$

$\frac{4n^3}{n^3} \quad \frac{64n^3}{n^3} \quad 64$

$2^n \to 2^{4n} = 2^{4n-n}$

$= 2^{3n} = (2^3)^n = 8^n$

Q4: For each of the following functions, indicate the class $\Theta(g(n))$ the function belongs to. (Use the simplest $g(n)$ possible in your answers.) Prove your assertions.

a. $(n^2+1)^{10}$ $T(n) = n^{20}$ $g(n^{20}) \to \Theta(n^{20})$

b. $\sqrt{10n^2+7n+3}$ $\sqrt{10n^2}+\sqrt{7n}+\sqrt{3} = \sqrt{10}\,n+\sqrt{7n}+\sqrt{3} = g(n)$ $\Theta(n)$

c. $2n \lg(n+2)^2 + (n+2)^2 \lg \frac{n}{2}$

d. $2^{n+1} + 3^{n-1}$

e. $\lfloor \log_2 n \rfloor$

Q5: List the following functions according to their order of growth from the lowest to the highest:

(7) (1) (6) (4) (2) (3) (5)

$(n-2)!,\ 5\lg(n+100)^{10},\ 2^{2n},\ 0.001n^4+3n^3+1,\ \ln^2 n,\ \sqrt[3]{n},\ 3^n.$

$n!$ $\log(n)$ 4^n n^4 $\log^2 n$ $n^{(1/3)}$ 3^n

Q6: The *range* of a finite nonempty set of n real numbers S is defined as the difference between the largest and smallest elements of S. For each representation of S given below, describe in English an algorithm to compute the range. Indicate the time efficiency classes of these algorithms using the most appropriate notation (O, Θ, or Ω).

i. An unsorted array $\Theta(n)$

ii. A sorted array $\Theta(1)$

iii. A sorted singly linked list $\Theta(n)$

iv. A binary search tree $\Theta(\log n)$

range = max - min

worst Exact best

Q7: Use the informal definitions of O, θ, Ω and to determine whether the following assertions are true or false.

a. $n(n+1)/2 \in O(n^3)$ b. $n(n+1)/2 \in O(n^2)$

c. $n(n+1)/2 \in \Theta(n^3)$ d. $n(n+1)/2 \in \Omega(n)$

c) $\frac{n^2+n}{2} \in \Theta(n^3)$ false should be exact

d) $\frac{n^2+n}{2} \in \Omega(n)$ TRUE

a) $n(n+1)/2$

$\frac{n^2+n}{2} \in O(n^2)$ TRUE

b) $n(n+1)/2 \in O(n^2)$ TRUE

Question No. 4

c) $2n \log (n+2)^2 + (n+2)^2 \log n/2$

$= 4n \log (n+2) + (n+2)^2 [\log n - \log 2]$

$= \theta(n \log n) + \theta(n^2 \log n)$

d) $2^{n+1} + 3^{n-1} = \theta(2^n) + \theta(3^n)$

$= 2^n + 3^n$

$= \theta(2^n) + \frac{\theta 3^n}{3}$

e) $|\log_2 n| = \theta(\log_2 n)$

Appendix:

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

linear

Sorting

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n). \\ c & \text{implies that } t(n) \text{ has the same order of growth as } g(n). \\ \infty & \text{implies that } t(n) \text{ has a larger order of growth than } g(n). \end{cases}$$

$T(n) = 4n^3 + n^2 + n$

$g(n^3) \rightarrow$ groth of n^3

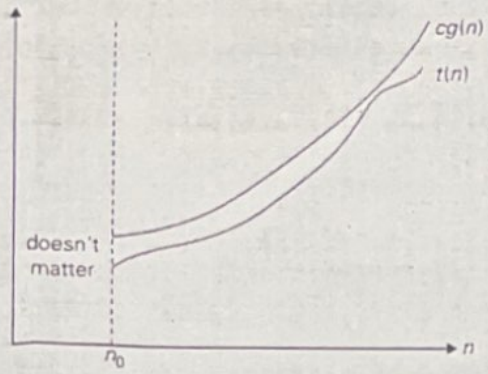
Question 1:-

i.

```
Max ← 0   min ← ∞
for(i ← 0 to n-1)
    if A[i] < min
        min ← A[i]
    if A[i] > max
        max ← A[i]
```

TABLE 2.2 Basic asymptotic efficiency classes

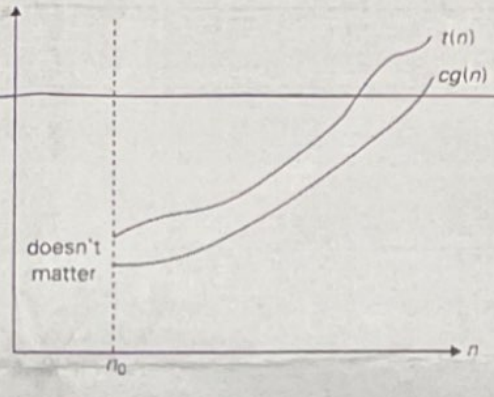
Class	Name	Comments
1	*constant*	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	*logarithmic*	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 4.4). Note that a logarithmic algorithm cannot take into account all its input or even a fixed fraction of it: any algorithm that does so will have at least linear running time.
n	*linear*	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.
$n \log n$	*linearithmic*	Many divide-and-conquer algorithms (see Chapter 5), including mergesort and quicksort in the average case, fall into this category.
n^2	*quadratic*	Typically, characterizes efficiency of algorithms with two embedded loops (see the next section). Elementary sorting algorithms and certain operations on $n \times n$ matrices are standard examples.
n^3	*cubic*	Typically, characterizes efficiency of algorithms with three embedded loops (see the next section). Several nontrivial algorithms from linear algebra fall into this class.
2^n	*exponential*	Typical for algorithms that generate all subsets of an n-element set. Often, the term "exponential" is used in a broader sense to include this and larger orders of growth as well.
$n!$	*factorial*	Typical for algorithms that generate all permutations of an n-element set.



$\in O(n^5)$
$\in O(n^4)$
$\in O(n^3)$
$\notin O(n)$
$\notin O(n \log n)$

worst O

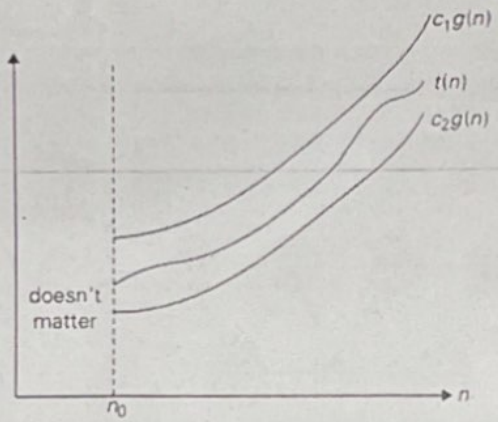
FIGURE 2.1 Big-oh notation: $t(n) \in O(g(n))$.



$\notin \Omega(n^4)$
$\in \Omega(n^3)$
$\in \Omega(n^2)$
$\in \Omega(n)$

best Ω

FIGURE 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$.



$\notin (n^4)$
$\in (n^3)$
$\notin (n^2)$

Exact Θ

FIGURE 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$.

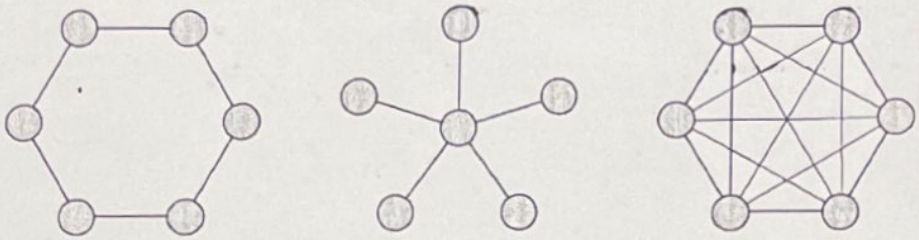
$T(n) = 5n^3 + n^2$

$g(n^3)$

sec sheet 5

Sheet 4 (Ch3)

Q1: A network topology specifies how computers, printers, and other devices are connected over a network. The figure below illustrates three common topologies of networks: the ring, the star, and the fully connected mesh.



ring star fully connected mesh

You are given a Boolean matrix $A[0..n-1, 0..n-1]$, where $n > 3$, which is supposed to be the adjacency matrix of a graph modeling a network with one of these topologies. Your task is to determine which of these three topologies, if any, the matrix represents. Design a brute-force algorithm for this task and indicate its time efficiency class.

Q2: Determine the number of character comparisons made by the brute-force algorithm in searching for the pattern **GANDHI** in the text: **THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED**
Assume that the length of the text—it is 47 characters long—is known before the search starts.

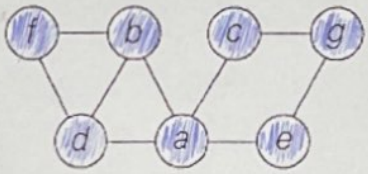
Q3: Let $x_1 < x_2 < \ldots < x_n$ be real numbers representing coordinates of n villages located along a straight road. A post office needs to be built in one of these villages.

a. Design an efficient algorithm to find the post-office location minimizing the average distance between the villages and the post office.

b. Design an efficient algorithm to find the post-office location minimizing the maximum distance from a village to the post office.

Q4: Consider the ***clique problem***: given a graph G and a positive integer k, determine whether the graph contains a ***clique*** of size k, i.e., a complete subgraph of k vertices. Design an exhaustive-search algorithm for this problem.

Q5: Consider the following graph.



a. Write down the adjacency matrix and adjacency lists specifying this graph.
(Assume that the matrix rows and columns and vertices in the adjacency lists follow in the alphabetical order of the vertex labels.)

b. Starting at vertex a and resolving ties by the vertex alphabetical order, traverse the graph by depth-first search and construct the corresponding depth-first search tree. Give the order in which the vertices were reached for the first time (pushed onto the traversal stack) and the order in which the vertices became dead ends (popped off the stack).

Sec 4

Sheet 3 (Ch2: 2.3→2.5)

Q1: Compute the following sums.

a. $1+3+5+7+\cdots+999$

b. $2+4+8+16+\cdots+1024$

c. $\sum_{i=3}^{n+1} 1$ d. $\sum_{i=3}^{n+1} i$ e. $\sum_{i=0}^{n-1} i(i+1)$

f. $\sum_{j=1}^{n} 3^{j+1}$ g. $\sum_{i=1}^{n}\sum_{j=1}^{n} ij$ h. $\sum_{i=1}^{n} 1/i(i+1)$

Q2: Consider the following algorithm.

ALGORITHM *Mystery(n)*
//Input: A nonnegative integer n
$S \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
$S \leftarrow S + i * i$
return S

a. What does this algorithm compute?
b. What is its basic operation?
c. How many times is the basic operation executed?
d. What is the efficiency class of this algorithm?
e. Suggest an improvement, or a better algorithm altogether, and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

Q3: Consider the following recursive algorithm.

ALGORITHM Q(n)
//Input: A positive integer n
if n = 1 **return** 1
else return Q(n − 1) + 2 • n − 1

a. Set up a recurrence relation for this function's values and solve it to determine what this algorithm computes.
b. Set up a recurrence relation for the number of multiplications made by this algorithm and solve it.
c. Set up a recurrence relation for the number of additions/subtractions made by this algorithm and solve it.

Q4: Consider the following recursive algorithm.

ALGORITHM *Riddle*$(A[0..n-1])$
//Input: An array $A[0..n-1]$ of real numbers
if $n = 1$ **return** $A[0]$
else $temp \leftarrow$ *Riddle*$(A[0..n-2])$
if $temp \le A[n-1]$ **return** $temp$
else return $A[n-1]$

a. What does this algorithm compute?
b. Set up a recurrence relation for the algorithm's basic operation count and solve it.

Q5: Fibonacci's rabbits problem A man put a pair of rabbits in a place surrounded by a wall. How many pairs of rabbits will be there in a year if the initial pair of rabbits (male and female) are newborn and all rabbit pairs are not fertile during the first month of life but thereafter give birth to one new male/female pair at the end of every month?