Grading policy

- **2 exam = 35 grades**
- **2 Quizzes = 5 grades**
- **Final exam = 60 grades.**

Total =100

Course materials

- **Lectures notes.**
- **Text books:**
  - **ALGORITHMSDESIGN TECHNIQUES AND ANALYSI M. H. Alsuwaiyel Information & Computer Science Department KFUPM July, 1999**
  - **Lectures of Algorithm Design And Analysis Asst. Prof. Ali Kadhum Idrees -Babylon University- College of Science for Women**
  - **Many other resources will appear throw course**

The Algorithm:

An algorithm is a finite set of instructions that if followed accomplish particular task. In addition, every algorithm must satisfy the following criteria:

1- Inputs: there are zero or more inputs which are externally supplied.

2- Outputs: At least one output is produced.

3- Definiteness: each instruction in algorithm must be clear and unambiguous, example for clear instruction (add 6 to x) and example for ambiguous instruction (divide 5 on 0), i.e., all people must understand it at the same the understanding and solve it at the same the solving.

4- Finiteness: If we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after la finite number of steps.

5- Effectiveness: every instruction must be sufficiently basic that it can, in principle, be carried out by a person using only pencil and paper. It is not enough that each operation be definite as in (3), but it must also be feasible.

We can distinguish between an algorithm and a program as follow:

Each algorithm must contain the previous five conditions and can be described in many ways:

• A natural language such as English can be used but we must be very careful that the resulting instructions are definite (condition 3).

• Pseudo-code.

• Flow chart.

A program does not necessarily satisfy condition 4. One important example of such a program for a computer is its operating system, which never terminate (except for system crashes) but continue in a wait loop until more jobs are entered. We will deal strictly with programs that always terminate.

Performance Analysis of Algorithm:

Algorithm analysis is the determining of algorithm efficiency and then enhancing it. There is two criterions directly relates to algorithm performance that are:

1-Space Complexity:

The space complexity of a program is the amount of memory it needs to run to completion. The space needed by each program is seen to be the sum of the following components:

• A Fixed Space Requirements (C)

Independent of the characteristics of the inputs and outputs

- instruction space
- space for simple variables, fixed-size structured variable, constants

• A Variable Space Requirements ($S_P(I)$)

Depend on the instance characteristic I

- number, size, values of inputs and outputs associated with I
- recursive stack space, formal parameters, local variables, return address

The space requirement *S*(p) of any program p may therefore be written as:

$S(p) = C + Sp$ (Instance Characteristics) where c is constant.

2-Time Complexity:

The time complexity of a program is the amount of computer time it needs to run to completion. The time, T(p), taken by a program p, is the sum of the compile time and the run (or execution) time.

The compile time does not depend on the instance characteristics. Also, we may assume that a compiled program will be run several times without recompilation. Consequently, we shall concern ourselves with just the run time of the program.

The run time is defined by *t*p (Instance Characteristics). The time requirement *T*(p) of any program p may therefore be written as:

T(p) = c + *t*p (Instance Characteristics) where c is the compilation time.

Two manageable approaches to estimating run time are (1) identify one or more key operations and determine the number of times these are performed and (2) determine the total number of steps executed by the program.

Methods to compute the step count

1-Step Count Method

***Program**	**Cost**	**Time**
float sum(float list[ ], int n)		
{		
float tempsum = 0;	C1	1
int i;		
for (i = 0; i < n; i++)	C2	n+1
count += 2;	C3	n
count += 3;	C4	1
return 0;	C5	1
}		

Total :C1+C2(n+1)+C3(n)+C4+C5= C1+C2n+C2+C3n+ C4+C5

$S_{sum}(I) = 0$

2-Tabular Method

Statement	s/e	Frequency	Total steps
float sum(float list[ ], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for(i=0; i <n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
Total			2n+3

$S_{sum}(I) = 0$