GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)
Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

on

DATA STRUCTURES

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth

Program Educational Objectives

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations
- Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the publiC)

HANDOUT ON DATA STRUCTURES

Class & Sem. : II B. Tech – I Semester Year : 2019-20

Branch : CSE Credits: 3

==========

1. Brief History and Scope of the Subject

A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for implementation of databases, while compiler implementations usually use hash tables to look up identifiers. Data structures are used in almost every program or software system. Data structures provide a means to manage huge amounts of data efficiently, such as large databases and internet indexing services. Usually, efficient data structures are a key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design.

2. Pre-Requisites

- Knowledge of any programming language that supports pointers for referencing.

3. Course Objectives:

- To gain knowledge of linear and non-linear data structures.
- To familiarize with different sorting and searching techniques

4. Course Outcomes:

Upon successful completion of the course, the students will be able to

- demonstrate the working process of sorting (bubble, insertion, selection and heap) and searching (linear and binary) methods using a programming language.
- design algorithms to create, search, insert, delete and traversal operations on linear and non-linear data structures.
- evaluate the arithmetic expressions using stacks.

- choose appropriate collision resolution techniques to resolve collisions.
- compare array and linked list representation of data structures.

5. Program Outcomes:

Graduates of the Computer Science and Engineering Program will have an ability to

a. apply knowledge of computing, mathematics, science and engineering fundamentals to solve complex engineering problems.

b. formulate and analyze a problem, and define the computing requirements appropriate to its solution using basic principles of mathematics, science and computer engineering.

c. design, implement, and evaluate a computer based system, process, component, or software to meet the desired needs.

d. design and conduct experiments, perform analysis and interpretation of data and provide valid conclusions.

e. use current techniques, skills, and tools necessary for computing practice.

f. understand legal, health, security and social issues in Professional Engineering practice.

g. understand the impact of professional engineering solutions on environmental context and the need for sustainable development.

h. understand the professional and ethical responsibilities of an engineer.

i. function effectively as an individual, and as a team member/ leader in accomplishing a common goal.

j. communicate effectively, make effective presentations and write and comprehend technical reports and publications.

k. learn and adopt new technologies, and use them effectively towards continued professional development throughout the life.

l. understand engineering and management principles and their application to manage projects in the software industry.

6. Mapping of Course Outcomes with Program Outcomes:

	a	b	c	d	e	f	g	h	i	i	k
CO1		3									
CO2		3									
CO3		3									
CO4			3		2						
CO5					2						

7. Prescribed Text Books

a. Debasis samanta, Classic Data Structures, PHI, 2nd edition, 2011.

b. Richard F, Gilberg , Forouzan, Data Structures, 2nd edition, , Cengage

8. Reference Text Books

a. Seymour Lipschutz, Data Structure with C, TMH.

b. G. A. V. Pai, Data Structures and Algorithms, TMH, 2008.

c. Horowitz, Sahni, Anderson Freed, Fundamentals of Data Structure in C, University Press, 2nd edition

9. URLs and Other E-Learning Resources

a. https://www.courserA)org/learn/data-structures
b. http://www.studytonight.com/data-structures/
c. http://www.indiabix.com/technical/data-structures/
d. http://nptel.aC)in/courses/106102064/1
e. http://freevideolectures.com/Course/2279/Data-Structures-And-Algorithms/2#

10. Lecture Schedule / Lesson Plan

Topic	No. of Periods	
	Theory	Tutorial
UNIT – 1: Searching and Sorting		
Concepts of data structures, Overview of data structures	1	1
Linear search	1	
Binary search	1	
Internal sorting: Basic concept	1	1
Bubble sort	1	
Insertion sort	1	
Selection sort	1	
	7	**2**
UNIT –2: Linked Lists		
Linked Lists- Basic concepts	1	1
Single linked list-operations	4	
Circular linked list	2	
Double linked list	4	1
	11	**2**
UNIT – 3: Stacks and Queues		
Stack introduction, Array and Linked List representations of stack	2	1
Operations on stacks using array and linked list	4	
Evaluation of arithmetic expression	3	
Queue introduction, Array and Linked List representations of queue	2	1
Operations on queues using array and linked list	3	
Circular queue introduction	1	
	15	**2**
UNIT – 4: Trees		
Basic tree concepts, Properties	2	1
Representation of Binary Trees using Arrays, linked lists	1	
Binary Tree Traversals (recursive)	1	
Binary search trees: Basic concepts, Search, insertion operations	2	1
Deletion Operation (Examples only)	1	
Creation of binary search tree from in-order and pre (post) order	1	
	8	**2**
UNIT – 5: Heap Trees and Graphs		
Heap Trees: Basic Concept, Operations	2	1
Graphs-Basic concepts, Representations of graphs	2	1
Graph traversals Breadth First Search (BFS), Depth	4	

First Search (DFS)		
	8	**2**
UNIT - 6: Hashing		
Hashing: Basic concepts, hashing functions (division method, multiplication method)	3	1
Collision resolution techniques- open hashing	1	1
Closed hashing (Linear Probing, Quadratic Probing, Double Hashing)	3	
	7	**2**
Total Number of Hours	**56**	**12**

11. Seminar Topics: -

UNIT – I

Sorting and Searching

Objective:

- To impart the concepts of searching and sorting.

Syllabus:

Sorting and Searching

Introduction- Concept of data structures, overview of data structures.

Searching: Linear search, Binary search.

Sorting (Internal): Basic concepts, sorting by: insertion (insertion sort), selection (selection sort), exchange (bubble sort).

Introduction to external searching and sorting

Learning Outcomes:

At the end of the unit student will be able to:

- demonstrate the working process of sorting (bubble, insertion, selection and heap) and searching (linear and binary) methods using a programming language.

Learning Material

Searching:

Is a process of verifying whether the given element is available in the given set of elements or not. Types of Searching techniques are:

1. Linear Search
2. Binary Search
3. Fibonacci Search

1. Linear Search

In linear search, search process starts from starting index of array. i.e. **0^{th}** index of array and end's with ending index of array. i.e. **$(n-1)^{th}$** index. Here searching is done in Linear fashion (Sequential fashion).

Algorithm Linearsearch(a<array>, n, ele)

Input: ***a*** is an array with *n* elements, *ele* is the element to be searcheD)

Output: Position of required element in array, if it is available.

1. found = 0
2. i =0
3. while(i < n)
    a) if(ele == a[i])

i) found = 1

ii) print(element found at i^{th} position)

iii) break

b) end if

c) i = i +1

4. end loop
5. if(found == 0)
   a) print(required element to be search is not available)
6. end if

End Linearsearch

Algorithm Linearsearch_Recurssion(a<array>, i, n, ele)

Input: *a* is an array with n elements, *ele* is the element to be searched, *i* is starting index of array and *n* is the ending index.

Output: Position of required element in array, if it is available.

1. found = 0
2. if(i<n)
   a) if(a[i] == ele)
      i) found = 1
      ii) print(element found at i^{th} position)
   b) end if
   c) Linearsearch_Recurssion(a, i+1, n, ele)
4. end if
5. if(found == 0)
   a) print(required element to be search is not available)
6. end if

End Linearsearch_Recurssion

2. Binary Search

The input to binary search must be in *ascending order*. i.e. set of elements be in ascending order.

Searching process in Binary search as follows:

- First, the element to be search is compared with middle element of array.

- If the required element to be searched is equal to middle element of array then Successful Search.
- If the required element to be searched is *less than* the middle element of array, then search in LEFT side of the midpoint of the array.
- If the required element to be search is *greater than* middle element of array, then search in RIGHT side of the midpoint of the array.

Algorithm Binarysearch(a<array>, n, ele)

Input: *a* is an array with *n* elements, *ele* is the element to be searcheD)

Output: Position of required element in array, if it is available.

1. found = 0
2. low = 0
3. high = n-1
4. while(low <= high)
    a) mid = (low+high)/2.
    b) if(ele == a[mid])
        i) print(required element was found at mid position)
        ii) found = 1
        iii) break
    c) else if(ele < a[mid])
        i) high = mid - 1
    d) else if(ele > a[mid])
        i) low= mid + 1
    e) end if
5. end if
6. if(found == 0)
    a) print(required element is not available)
7. end if

End Binarysearch

Algorithm Binarysearch_Recursion(a<array>,ele, low, high)

Input: ***a*** is array with n elements, *ele* is the element to be searched, *low* is starting index, *high* is ending index of array.

Output: Position of required element in array, if it is available.

1. found = 0
2. if(low <= high)
    a) mid = (low+high)/2.
    b) if(ele == a[mid])
        i) print(required element was found at mid position)
        ii) found = 1
    c) else if(ele < a[mid])
        i) Binarysearch _Recursion(a,ele,low,mid-1)
    d) else if(ele > a[mid])
        i) Binarysearch_Recursion(a,ele,mid+1,high)
    e) end if
3. end if
4. if(found == 0)
    a) print(required element to be search is not available)
5. end if

End Binarysearch_Recursion

Sorting: Sorting means arranging the elements either in ascending or descending order.

There are two types of sorting.

1. Internal Sorting.
2. External Sorting.

1. Internal Sorting: For sorting a set of elements, if we use only primary memory (Main memory), then that sorting process is known as internal sorting. i.e. internal sorting deals with data stored in computer memory.

2. External Sorting: For sorting a set of elements, if we use both primary memory (Main memory) and secondary memory, then that sorting process is known as external sorting. i.e. external sorting deals with data stored in files.

Different types of sorting techniques.

1. Bubble sort
2. Insertion sort
3. Selection sort
4. Merging sort
5. Quick sort
6. Radix sort

1. Bubble sort:

- In bubble sort for sorting n elements, we require (n-1) passes (or) iterations and in each pass compare every element with its successor. i.e. i^{th} index element will compare with $(i+1)^{th}$ index element, if they are not in ascending order, then swap them.
- Here for each pass, the largest element is moved to height index position of array to be sort.

Process:

1. In pass1, a[0] and a[1] are compared, then a[1] is compared with a[2], then a[2] is compared with a[3] and so on. Finally a[n-2] is compared with a[n-1]. Pass1 involves (n-1) comparisons and places the biggest element at the highest index of the array to be sorteD)
2. In pass2, a[0] and a[1] are compared, then a[1] is compared with a[2], then a[2] is compared with a[3] and so on. Finally a[n-3] is compared with a[n-2]. Pass2 involves (n-2) comparisons and places the biggest element at the highest index of the array to be sorteD)
3. In pass (n-1), a[0] and a[1] are compareD) After this step all the elements of the array are arranged in ascending order.

Eg: sort the elements 72, 85, 4, 32 and 16 using Bubble sort.

	0	**1**	**2**	**3**	**4**	
Pass 1	72	85	24	32	16	
	72	85	24	32	16	(0, 1) No Exchange
	72	24	85	32	16	(1, 2) Exchange
	72	24	32	85	16	(2, 3) Exchange
	72	24	32	16	**85**	(3, 4) Exchange
Pass 2	72	24	32	16	85	
	24	72	32	16	85	(0, 1) Exchange
	24	32	72	16	85	(1, 2) Exchange
	24	32	16	**72**	**85**	(2, 3) Exchange

Pass 3	24	32	16	72	85	
	24	32	16	72	85	(0, 1) No Exchange
	24	16	**32**	**72**	**85**	(1, 2) Exchange
Pass 4	24	16	32	72	85	
	16	**24**	**32**	**72**	**85**	(0, 1) Exchange
Sorted List is	**16**	**24**	**32**	**72**	**85**	

Algorithm Bubblesort(a<array>, n)

Input: *a* is an array with n elements to be sort.

Output: array elements in ascending order.

1. for(i = 0 to n-1)
    a) for(j = 0 to n-i-1)
        i) if (a[j] > a[j+1])
            A) t = a[j]
            B) a[j] = a[j+1]
            C) a[j+1] = t
        ii) end if
    b) end j for loop
2. end i for loop

End Bubblesort

2. Insertion sort

- In the insertion sort, initially consider the 0^{th} index element value as only sorted element, and then take remaining elements of the given set one by one.
- For every pass, compare unsorted elements one by one with sorted list.
- If sorted list value is GREATER than the unsorted element value, then **move** sorted list element to **next index.**
- Continue the above moving process up to sorted list element value is LESS than given unsorted element value.
- Continue the above process for all the elements in the given array.

Algorithm insertionsort(a<array>, n)

Input: a is array with n elements to be sorteD)

Output: array elements in ascending order.

1. i = 1
2. while(i < n)
   a) x = a[i] /* x is unsorted element */
   b) j = i -1
   c) while(j >= 0 && a[j] > x)
      i) a[j+1] = a[j]
      ii) j = j -1
   d) end loop
   e) a[j+1] = x
   f) i = i + 1
3. end loop

End insertionsort

Eg: sort the elements 15,10, 8, 46, 32 using Insertion sort. Where **x** is an unsorted element

	0	**1**	**2**	**3**	**4**		**x**
Pass 1	15	10	8	46	32	Sorted *ele* > unsorted *ele*. TRUE. i.e. **15>10**. So move 15 from 0^{th} index to 1^{st} index	10
		15	8	46	32		
	10	15	8	46	32	Last index is 0^{th} index. So place **x** value in 0^{th} index	

	0	**1**	**2**	**3**	**4**		**x**
Pass 2	**10**	**15**	8	46	32	Sorted *ele* > unsorted *ele*. TRUE. i.e. 15 > 8 So move 15 from 1^{st} index to 2^{nd} index	8
	10		15	46	32	Sorted *ele* > unsorted *ele*. TRUE. i.e. 10 > 8 So move 10 from 0^{th}index to 1^{st}index	
		10	15	46	32		
	8	**10**	**15**	46	32	Last index is 0^{th} index. So place **x** value in 0^{th} index	

	0	**1**	**2**	**3**	**4**		**x**
Pass 3	**8**	**10**	**15**	46	32	Sorted *ele*> unsorted *ele*. FALSE. i.e. 15>46 is FALSE	46

						So place x value in next index of sorted element	
	8	**10**	**15**	**46**	32		
Pass 4	**0**	**1**	**2**	**3**	**4**		**x**
	8	**10**	**15**	**46**	32	Sorted *ele*> unsorted *ele*. TRUE. i.e. 46 > 32 So move 46 from 3rd index to 4th index	32
	8	**10**	**15**		**46**	Sorted *ele*> unsorted *ele*. FALSE. i.e. 15 >32 is FALSE So place x value in next index of sorted element	
	8	**10**	**15**	**32**	**46**		

Now all the elements are sorted

3. Selection Sort

In selection sort first find the smallest element in the array and place it in to the array to the 0^{th} position. Then find the second smallest element in the array and place it in 1^{st} position. Repeat this procedure until array is sorteD)

Process:

- In Pass1, find the position of the smallest element in the array and then swap a[pos] and a[0]. Now a[0] is sorteD)
- In Pass2, find the position of the smallest element in sub array of n-1 elements, then swap a[pos] and a[1]. Now a[1] is sorteD)
- In Pass n-1, find the position of the smallest element from a[n-2] and a[n-1], then swap a[pos] and a[n-21]. So that a[0], a[1], a[2], a[3], ..., a[n-1] is sorteD)

Eg: sort the elements 15, 10, 11, 41, 3 using selection sort

	0	**1**	**2**	**3**	**4**		pos
Pass 1	15	10	11	41	3	pos is initially assigned at 0^{th} index	0
	15	10	11	41	3	a[1] < a[pos] is TRUE. So pos =1	1
	15	10	11	41	3	a[2] < a[pos] is FALSE. So No change in pos	1
	15	10	11	41	3	a[3] < a[pos] is FALSE. So No change in pos	1
	15	10	11	41	3	a[4] < a[pos] is TRUE. So pos =4	4

Now swap a[pos] and a[0]

0	**1**	**2**	**3**	**4**
3	10	11	41	**15**

Now a[0] is sorted

	0	**1**	**2**	**3**	**4**		**pos**
Pass 2	**3**	10	11	41	15	Now pos is assigned at 1st index	1
	3	10	11	41	15	a[2] < a[pos] is FALSE. So No change in pos	1
	3	10	11	41	15	a[3] < a[pos] is FALSE. So No change in pos	1
	3	10	11	41	15	a[4] < a[pos] is FALSE. So No change in pos	1

Now swap a[pos] and a[1]

0	**1**	**2**	**3**	**4**
3	**10**	11	41	15

Now a[1] is sorted

	0	**1**	**2**	**3**	**4**		**pos**
Pass 3	**3**	**10**	11	41	15	Now pos is assigned at 2nd index	2
	3	**10**	11	41	15	a[3] < a[pos] is FALSE. So No change in pos	2
	3	**10**	11	41	15	a[4] < a[pos] is FALSE. So No change in pos	2

Now swap a[pos] and a[2]

0	**1**	**2**	**3**	**4**
3	**10**	**11**	41	15

Now a[2] is sorted

	0	**1**	**2**	**3**	**4**		**pos**
Pass 4	**3**	**10**	**11**	41	15	Now pos is assigned at 3rd index	3
	3	**10**	**11**	41	15	a[4] < a[pos] is TRUE. So pos = 4	4

Now swap a[pos] and a[3]

0	**1**	**2**	**3**	**4**
3	**10**	**11**	**15**	**41**

Now all the elements are sorted

Algorithm selectionsort (a<array>, n)

Input: *a* is an array with *n* elements to be sorteD)

Output: array elements in ascending order.

1. i = 0
2. while(i < n)
    a) pos = i
    b) j = i+1
    c) while (j < n)

i) if (a[j] < a[pos])

A) pos = j

ii) end if

iii) j = j +1

d) end loop

e) t = a[pos]

f) a[pos] = a[i]

g) a[i] =t

h) i= i+1

3. end loop

End selectionsort

UNIT-I
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. Find the location of the element with a given value is___?[ ]
   A) Traversal B) Searching C) Sorting D) None of above
2. Which of the following is false? [ ]
   A) A linear search begins with the first array element
   B) A linear search continues searching, element by element, either until a match is found or until the end of the array is encountered
   C) A linear search is useful when the amount of data that must be search is small
   D) For a linear search to work, the data in the array must be arranged in either alphabetical or numerical order
3. Which characteristic will be used by binary search but the linear search ignores is _____. [ ]
   A) Order of the elements of the list B) Length of the list
   C) Maximum value in list D) Type of elements of the list
4. Choose the false statement. [ ]
   A) A binary search begins with the middle element in the array
   B) A binary search continues having the array either until a match is found or until there are no more elements to search
   C) If the search argument is greater than the value located in the middle of the binary, the binary search continues in the lower half of the array
   D) For a binary search to work, the data in the array must be arranged in either alphabetical or numerical order

5. Which of the following is *not* a limitation of binary search algorithm? [ ]

   A) Must use a sorted array

   B) Requirement of sorted array is expensive when a lot of insertion and deletions are needed

   C) There must be a mechanism to access middle element directly

   D) Binary search algorithm is not efficient when the data elements more than 1500

6. What is the complexity of searching an element from a set of n elements using Binary search algorithm is [ ]

   A) O(n) B) O(log n) C) $O(n^2)$ D) O(n log n)

7. Label the process of arranging values in an ordered manner is called as _____.

8. In which sorting technique, consecutive adjacent pairs of elements in the array are compared with each other. [ ]

   A) Bubble sort B) Selection Sort C) Insertion Sort D) None

9. Identify the number of comparisons required to sort a list of *10* numbers in *pass 2* by using *Bubble Sort* is_____. [ ]

   A) 10 B) 9 C) 8 D) 7

11. Consider an array of elements arr[5]= {99,22,55,44,33}, what are the steps done while doing bubble sort in the array. [ ]

   A) 22 55 44 33 99 33 22 44 99 55 22 44 99 33 55 44 22 55 33 99

   B) 22 55 44 33 99 22 44 33 55 99 22 33 44 55 99 22 33 44 55 99

   C) 55 44 33 99 22 44 22 33 99 55 55 33 99 22 44 99 55 44 33 22

   D) None of the above

12. Which sorting technique sorts a list of elements by moving the current data element past the already sorted values with the preceding value until it is in its correct place. [ ]

A) Insertion sort B) Bubble Sort C) Selection Sort D) None

13. Identify the number of passes required by insertion sort for the list **size 15.** [ ]

A) 15 B) 16 C) 14 D) 13

14. Which of the following sorting algorithms in its implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced). [ ]

A) Insertion sort B) Bubble Sort C) Selection Sort D) None

15. Consider an array of elements arr[5]= {5,4,3,2,1} , what are the steps of insertions done while doing insertion sort in the array. [ ]

A) 4 5 3 2 1 3 4 5 2 1 2 3 4 5 1 1 2 3 4 5

B) 5 4 3 1 2 5 4 1 2 3 5 1 2 3 4 1 2 3 4 5

C) 4 3 2 1 5 3 2 1 5 4 2 1 5 4 3 1 5 4 3 2

D) 4 5 3 2 1 2 3 4 5 1 3 4 5 2 1 1 2 3 4 5

16. Consider the array A[]= {6,4,8,1,3} apply the *insertion sort* to sort the array . Consider the cost associated with each sort is 25 rupees, what is the total cost of the insertion sort when element 1 reaches the first position of the array? [ ]

A) 50 B) 25 C) 75 D) 100

17. Consider a situation where swap operation is very costly. Which of the following sorting algorithms should be preferred so that the numbers of swap operations are minimized in general? [ ]

A) Bubble Sort B) Selection Sort C) Insertion Sort D) None

18. Which one of the following in-place sorting algorithms needs the minimum number of swaps? [ ]

A) Insertion Sort B) Bubble Sort C) Selection Sort D) All of the above

19. Discover the comparisons needed to sort an array of length 5 if a straight selection sort is used and array is already in the opposite order? [ ]

A) 1 B) 10 C) 5 D) 20

20. Determine the advantage of bubble sort over other sorting techniques? [ ]

a) It is faster

b) Consumes less memory

c) Detects whether the input is already sorted

d) All of the mentioned

SECTION-B

SUBJECTIVE QUESTIONS

1. Given a telephone directory and a name of the subscriber, choose search method you would suggest for finding the telephone number of the given subscriber.
2. Apply linear search for an element 18 and 100 in the following list.

   36, 72, 19, 45, 18, 22, 12, 55
3. Apply binary search for an element 54 and 100 in the following list.

   13, 27, 91, 54, 81, 6, 51, 59, 45, 69
4. Make use of bubble sort for the following elements.

   30, 52, 29, 87, 63, 27, 19, 54
5. Make use of insertion sort for the following elements.

   59, 19, 54, 96, 81, 801, 45, 72, 64, 92
6. Make use of selection sort for the following elements.

   36, 12, 81, 45, 90, 27, 72, 18
7. Explain bubble sort algorithm.

8. Explain insertion sort algorithm.
9. Explain selection sort algorithm.
10. Explain non recursive linear search algorithm.
11. Explain recursive binary search algorithm.
12. Develop a C program using for loop to find all the occurrences of a given key in a given list using linear search. The algorithm should display locations of all the occurrences of the given key. Discuss with an example.

SECTION-C

QUESTIONS AT THE LEVEL OF GATE

1. Consider the C function given below. Assume that the array listA contains n (> 0) elements, sorted in ascending order.(**GATE-CS-2014)** **[ ]**

```
int ProcessArray(int *listA, int x, int n)
{
    int i, j, k;
    i = 0;
    j = n-1;
    do
    {
        k = (i+j)/2;
        if (x <= listA[k])
            j = k-1;
        if (listA[k] <= x)
            i = k+1;
    } while (i <= j);
```

```
        if (listA[k] == x)
                return(k);
        else
                return -1;
}
```

Which one of the following statements about the function ProcessArray is CORRECT?
(A) It will run into an infinite loop when x is not in listA.
(B) It is an implementation of binary search.
(C) It will always find the maximum element in listA.
(D) It will return −1 even when x is present in listA.

2. Consider the following C program that attempts to locate an element x in an array Y[ ] using binary search. The program is erroneous. **(GATE CS 2008)** **[ ]**

```
1.  f(int Y[10], int x) {
2.    int i, j, k;
3.    i = 0; j = 9;
4.    do {
5.          k =  (i + j) /2;
6.          if( Y[k] < x)  i = k; else j = k;
7.       } while(Y[k] != x && i < j);
8.    if(Y[k] == x) printf ("x is in the array ") ;
9.    else printf (" x is not in the array ") ;
10. }
```

On which of the following contents of Y and x does the program fail?
(A) Y is [1 2 3 4 5 6 7 8 9 10] and x < 10
(B) Y is [1 3 5 7 9 11 13 15 17 19] and x < 1
(C) Y is [2 2 2 2 2 2 2 2 2 2] and x > 2
(D) Y is [2 4 6 8 10 12 14 16 18 20] and 2 < x < 20 and x is even

3. In the above question, the correction needed in the program to make it work properly is **(GATE CS 2008)** **[ ]**
(A) Change line 6 to: if (Y[k] < x) i = k + 1; else j = k-1;
(B) Change line 6 to: if (Y[k] < x) i = k – 1; else j = k+1;
(C) Change line 6 to: if (Y[k] <= x) i = k; else j = k;
(D) Change line 7 to: } while ((Y[k] == x) && (i < j));

4. The average number of key comparisons done in a successful sequential search in a list of length it is (**GATE CS 1996**)**[ ]**

(A) log n **(B)** (n-1)/2 **(C)** n/2 **(D)** (n+1)/2