COMSATS University, Islamabad
Islamabad Campus
Department of Computer Science

Read before Attempt

Assignment No. 4: Searching, Sorting, and Hashing	
Course code and Title: CSC211, Data Structure	
Instructor:	Atique Ahmed
Assigned Date: Dec 04, 2025	**Due Date: Dec 09, 2025**
Total Marks: --	
CLO-3(Theory): Apply fundamental sorting, searching, and hashing techniques on different data structures. **CLO -4(Lab): Implement different data structures, searching, sorting, and hashing in a programming language.**	

Performance Indicators for Every Assignment (Grading rubric)

Each assignment will be assigned a letter grade of either A, B, C, D, or F based on its submission.

This five-point (discrete) scale is described as follows:

Grade	Status	Description
A	Exemplary (=60-70%)	Solution presented solves the problem stated correctly and meets all require ments of the problem.
B	Capable (=50-60%)	Solution is mostly correct, satisfying most of the above criteria under the exemplary category, but contains some *minor* pitfalls, errors/flaws or limit ations.
C	Needs Improvement (=40-50%)	Solution demonstrates a viable approach toward solving the problem but contains some *major* pitfalls, errors/flaws or limitations.
D	Unsatisfactory (=20-40%)	-Critical elements of the solution are missing or significantly flawed and does not demonstrate sufficient understanding of the problem.
F	Not attempted	Late/no submission

To discourage copying/cheating, 30% marks may be awarded based on the marks obtained either in the corresponding quiz or a question(s) in the Midterm exam that will be used as multiplying factor. Hence, the final grades (out of 100) will be calculated as follows:

Submission marks (given by the letter grades shown above) + 30 * multiplying factor

Question # 1

Consider the following two versions of selection sorts algorithm

ALGORITHM SelectionSort1(A[0-----n-1])

```
// Input: An array A[0-----n-1] of orderable elements
// Output: Array A[0----n-1] arranged in ascending order
for i ← 0 to n-2 do
    min ← i
    for j ← i+1 to n-1 do
        if A[j] < A[min] then
            min ← j
        end
    swap(A[i],A[min])
    end
end
```

ALGORITHM SelectionSort2 (A[1……n]

```
j = n
for i ← 1 to n-1 do
    [ max, min] ← MinMax(A[i…..j])
    swap(A[i], A[min])
    swap(A[j], A[max])
    j--
end
```

where MinMax is given below:

```
Algorithm MinMax (A[0..n − 1], minval, maxval)
// Input: An array A [0----n-1]) of n real numbers.
// Output: maxval – minval
minval ← A[0]; maxval ← A[0]
for i ← 1 to n-1 do
    if A[i] < minval then
        minval ← A[i]
    end
    if A[i] > maxval then
        maxval ← A[i]
    end
end
return ( maxval , minval )
```

---

a) Trace(dry run) above algorithms for the following inputs. Show your work step by step.
   i. Input 1: [13,9,2,5,25,4,19,8].
   ii. Input 2: [2,4,5,8,9,13,19,25].
   iii. Input 3: [25,19,13,9,8,5,4,2].

b) **Compare** the above sorting algorithms with respect to:
   i. the number of elements swapped

ii. the number of comparisons

Algorithm	Input Type	the number of elements moved/swapped	the number of comparisons
SelectionSort1	Input 1		
	Input 2		
	Input 3		
SelectionSort2	Input 1		
	Input 2		
	Input 3		

Question # 2

Consider the following three versions of insertion sort.

ALGORITHM *InsertionSort*($A[0..n-1]$)
//Sorts a given array by insertion sort
//Input: An array $A[0..n-1]$ of n orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 1$ **to** $n-1$ **do**
    $v \leftarrow A[i]$
    $j \leftarrow i-1$
    while $j \geq 0$ **and** $A[j] > v$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow v$

ALGORITHM *InsertSort2*($A[0..n-1]$)
for $i \leftarrow 1$ **to** $n-1$ **do**
    $j \leftarrow i-1$
    while $j \geq 0$ **and** $A[j] > A[j+1]$ **do**
        swap($A[j], A[j+1]$)
        $j \leftarrow j-1$

```
BinaryInsertionSort (int a[], int n)
{
   int ins, i, j;
   int tmp;

   for (i = 1; i < n; i++) {
      ins = BinarySearch (a, 0, i, a[i]);
      if (ins < i) {
         tmp = a[i];
         for (j = i - 1; j >= ins; j--)
            a[j + 1] = a[j];
         a[ins] = tmp;
      }
   }
}
```

```
BinarySearch (int a[], int low, int high, int key)
{
   int mid;

   if (low == high)
      return low;

   mid = low + ((high - low) / 2);

   if (key > a[mid])
      return BinarySearch (a, mid + 1, high, key);
   else if (key < a[mid])
      return BinarySearch (a, low, mid, key);

   return mid;
}
```

What is the number of comparison and interchanges performed by "**the simple insertion sort**" **{first two algorithms}** and "**the insertion sort using binary search**"(BinaryInsertionSort) for the following array

i. A sorted array
ii. An array that is sorted in reverse order (that is, from largest to smallest)
iii. An array in which A[0], A[2], A[4],……… are the smallest elements and are in sorted order, and in which elements A[1],A[3], A[4],……… are the largest elements and are in reverse order.
iv. An array in which A[0] through A[ind](where ind = (n-1)/2) are the smallest elements and are sorted, and in which A[ind +1] through A[n-1] are the largest elements and are in reverse sorted order..
v. An array in which A[0], A[2], A[4],……. Are the smallest elements in sorted order, and in which A[1], A[3],A[5],……. Are the largest elements in sorted order?

Question # 3

Consider the following three versions of Bubble sorts.

```
ALGORITHM BubbleSort1(A[0------n -1])
// Input: An array A[0-----n-1] of orderable elements
// Output: Array A [0----n-1] arranged in ascending order
    Count ⟵ 0
    for i ⟵ 0 to n-2 do
        for j ⟵ 0 to n-2 - i do
            if A[j+1] < A[j] then
                swap(A[j],A[j+1])

            end
        end
    end
    return Count
```

```
ALGORITHM BubbleSort2(A[0------n -1])
// Input: An array A[0-----n-1] of orderable elements
// Output: Array A [0----n-1] arranged in ascending order
    Count1 ← 0
    Count ← n-1
    sflag ← true
    While sflag do
        Sflag ← false
        for j ← 0 to Count -1do
            if A[j+1] < A[j] then
                swap(A[j],A[j+1])
                Count1 ← Count1 +1
                sflag ← true
            end if
            Count ← Count -1
        end for
    end while
    return Count1
```

```
ALGORITHM BubbleSort3(A[0……..N-1])
// Input:
//Output:
L ← 2
R ← N
K ← N
Repeat until L≥ R
for J ← R down to L do
  if (A[J] < A[J - 1]) then
     Swap(A[J], A[J - 1])
     K ← J
  end
L ← K + 1
for J ← L to R do
  if (A[J] < A[J - 1]) then
     Swap(A[J], A[J - 1])
     K ← J
  end
R ← K - 1
```

a) Trace(dry run) above algorithms for the following inputs. Show your work step by step.

   i. Input 1: [13,9,2,5,25,4,19,8].
   ii. Input 2: [2,4,5,8,9,13,19,25].
   iii. Input 3: [25,19,13,9,8,5,4,2].

b) **Compare** the above sorting algorithms with respect to:

i. the number of elements swapped

ii. the number of comparisons

Algorithm	the number of elements moved/swapped		the number of comparisons	Comments
Bubble Sort 1	Input 1			
	Input 2			
	Input 3			
Bubble Sort 2	Input 1			
	Input 2			
	Input 3			
Bubble Sort 3	Input 1			
	Input 2			
	Input 3			

Question # 4

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
1  if p < r
2     then q ← ⌊(p+r)/2⌋
3          MergeSort (A, p, q)
4          MergeSort (A, q+1, r)
5          Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
1  if p < r
2     then q ← ⌊(p+r)/2⌋
3          MergeSort (A, p, q)
4          MergeSort (A, q+1, r)
5          Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

replace statement 2 with $q \leftarrow \lfloor (p+r)/3 \rfloor$

```
Algorithm 2: Bottom-Up Merge Sort
  Input: sequence S with n elements
  Output: sequence S sorted single array
  Bottom Up Merge Sort;
  if (n < 2) then
     for i = 1; i < n; i = i + i do
        for j = 0; j < n − i; j+ = i + i do
           if (n < j + i + i) then
              merge(A + j, A + j + i, A + n)
           else
              merge(A + j, A + j + i, A + j + i + i)
           end
        end
     end
     return
  end
```

```
MERGE-SORT( array A, int p, int r)
1 if (p < r) then
2      q1 ← n/3
3      q2 ←2(n/3)
4      MERGE-SORT(A, p, q1)           // sort A[p.. q1]
5      MERGE-SORT(A, q1 + 1, q2)   // sort A[q1 + 1.. q2]
6      MERGE-SORT(A, q2 + 1, r)     // sort A[q2 + 1..r]
7      MERGE(A, p, q1, q2, r)         // merge the three pieces
```

Trace the above algorithms for the following input

18,12,16,22,5,24,9

Question # 5

Compare the "***Linear Search***" and "***Binary search***" by filling the following table, showing the ***number of comparison*** needed either to "*find the value*" or to "*determine that the value is not in the array*", given the array of values. Write a paragraph comparing your answers.

14	27	95	12	26	5	33	15	9	99
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Values	Search data Values Sequentially	Search sorted Values Sequentially	Search sorted Values using Binary Search
16			
17			
14			
5			
99			
100			
0			

Question # 6

A grocery store wants to assign a unique storage location for items in its inventory system. Each item is identified by a 6-digit **Product ID**. The hash table has 100 slots. Use the **Digit Folding** method for hashing.

Task: Given the following Product IDs:341276, 123456, 987654, 561231

- Calculate the hash values using the **Digit Folding** method (sum the digits in groups of 2).
- If two items hash to the same location, resolve the collision using **Linear Probing**

Question # 7

A university uses student IDs to store records in a hash table. The hash table has 50 slots, and the **Mid-Square** method is used to generate hash values. The university also uses **Chaining** to resolve collisions.

Task: For the following Student IDs: 2345, 3123, 6542, 7123, 8901

- Calculate the hash values using the **Mid-Square** method (square the ID, extract the middle 2 digits).
- Show the resulting hash table with chains for collisions.

Question # 8

A hospital maintains a database of patient records. Each patient is assigned a unique ID. The hospital's hash table has **m = 13** slots, and the **Division Method** is used to calculate the hash value ($h(k) = k \bmod m$). To resolve collisions, **Double Hashing** is used with the second hash function $h2(k)=7-(k \% 7)$.

Task: For the following Patient IDs: 105, 212, 318, 440, 527

Calculate the hash values and demonstrate how **Double Hashing** resolves collisions. Show all steps.

Question # 9

A real estate company maintains records of property IDs in a hash table of size 11. The property IDs are hashed using the **Division Method**. To resolve collisions, **Quadratic Probing** is used.

Task: Insert the following Property IDs into the hash table: 21, 34, 55, 44,89

Demonstrate all steps of insertion using **Quadratic Probing**.

Question # 10

A social media platform uses hashing to manage hashtags. Each hashtag is hashed based on its ASCII values using $h(hashtag)=\sum(ASCII(characters)) mod 10$.

Hashtags #fun, #sun, #run, #funrun, #sunday are added. Collisions are resolved using separate chaining.

LAB Tasks

Implement all sorting related questions(1-4) given in theory assignments.