Output - Stack

\*\*\*STACK MENU\*\*\*

1. PUSH 2. POP 3. DISPLAY 4. QUIT

Enter choice: 1

ELEMENT TO PUSH: 5

\*\*\*STACK MENU\*\*\*

1. PUSH 2. POP 3. DISPLAY 4. QUIT

Enter choice: 1

ELEMENT TO PUSH: 2

\*\*\*STACK MENU\*\*\*

1. PUSH 2. POP 3. DISPLAY 4. QUIT

Enter choice: 1

ELEMENT TO PUSH: 6

\*\*\*STACK MENU\*\*\*

1. PUSH 2. POP 3. DISPLAY 4. QUIT

Enter choice: 2

POPPED ELEMENT: 6

\*\*\*STACK MENU\*\*\*

1. PUSH 2. POP 3. DISPLAY 4. QUIT

Enter choice: 3

TOP CURRENTLY: 1

ELEMENTS OF STACK S:

S[1]: 2

S[0]: 5

\*\*\*STACK MENU\*\*\*

1. PUSH 2. POP 3. DISPLAY 4. QUIT

Enter choice: 4

Output – Stack: Postfix

Infix -> Postfix.

Enter Infix Expression: a+b\*c

Sr. no. CHAR STACK EXPR

0 ' ' ( ''

1 'a' ( 'a'

2 '+' (+ 'a'

3 'b' (+ 'ab'

4 '\*' (+\* 'ab'

5 'c' (+\* 'abc'

6 ')' EMPTY 'abc\*+'

Postfix Expression: abc\*+

Input 'a': 1

Input 'b': 2

Input 'c': 3

Sr no | SYMBOL | POP() [Y, X] | PUSH() | STACK\_E[0 -> TOP\_E]

1 | a | | 1.00 | 1.00

2 | b | | 2.00 | 1.00 2.00

3 | c | | 3.00 | 1.00 2.00 3.00

4 | \* | 3.00 2.00 | 6.00 | 1.00 6.00

5 | + | 6.00 1.00 | 7.00 | 7.00

6 | # | 7.00 | | EMPTY

RESULT = 7.000000

Output – Stack: Prefix

Infix -> Prefix.

Enter Infix Expression: a+b\*c

Sr. no. CHAR STACK EXPR

0 ' ' ) ''

1 'c' ) 'c'

2 '\*' )\* 'c'

3 'b' )\* 'cb'

4 '+' )+ 'cb\*'

5 'a' )+ 'cb\*a'

6 '(' EMPTY 'cb\*a+'

Prefix Expression: +a\*bc

Input 'a': 1

Input 'b': 2

Input 'c': 3

Sr no | SYMBOL | POP() [X, Y] | PUSH() | STACK\_E[0 -> TOP\_E]

1 | c | | 3.00 | 3.00

2 | b | | 2.00 | 3.00 2.00

3 | \* | 2.00 3.00 | 6.00 | 6.00

4 | a | | 1.00 | 6.00 1.00

5 | + | 1.00 6.00 | 7.00 | 7.00

6 | # | 7.00 | | EMPTY

RESULT = 7.000000

Output – Queue

\*\*QUEUE OPERATIONS\*\*

1. Enqueue 2. Dequeue 3. Display 4. Exit

Choice: 1

Enter element to enqueue: 5

\*\*QUEUE OPERATIONS\*\*

1. Enqueue 2. Dequeue 3. Display 4. Exit

Choice: 1

Enter element to enqueue: 6

\*\*QUEUE OPERATIONS\*\*

1. Enqueue 2. Dequeue 3. Display 4. Exit

Choice: 2

Dequeued 5 Successfully

\*\*QUEUE OPERATIONS\*\*

1. Enqueue 2. Dequeue 3. Display 4. Exit

Choice: 3

CURRENT QUEUE:

[0] [1] [2]

\*\*\* 6 \*\*\*

\*\*QUEUE OPERATIONS\*\*

1. Enqueue 2. Dequeue 3. Display 4. Exit

Choice: 4

Output – Circular Queue

---- CIRCULAR QUEUE OPERATIONS ----

1. Encqueue 2. Decqueue 3. Display 4. Exit

Choice: 1

Enter element to enqueue: 5

Encqueued 5 Successfully

---- CIRCULAR QUEUE OPERATIONS ----

1. Encqueue 2. Decqueue 3. Display 4. Exit

Choice: 1

Enter element to enqueue: 6

Encqueued 6 Successfully

---- CIRCULAR QUEUE OPERATIONS ----

1. Encqueue 2. Decqueue 3. Display 4. Exit

Choice: 2

Decqueued 5 Successfully

---- CIRCULAR QUEUE OPERATIONS ----

1. Encqueue 2. Decqueue 3. Display 4. Exit

Choice: 3

CURRENT CIRCULAR QUEUE:

[0] [1] [2] [3]

\*\*\* 6 \*\*\* \*\*\*

---- CIRCULAR QUEUE OPERATIONS ----

1. Encqueue 2. Decqueue 3. Display 4. Exit

Choice: 4

Output – Double Ended Queue

---- DE-QUEUE OPERATIONS ----

1. PUSHDQ 2. POPDQ 3. INJECT 4. EJECT 5. DISPLAY 6. EXIT

Choice: 1

Enter element to enqueue: 5

---- DE-QUEUE OPERATIONS ----

1. PUSHDQ 2. POPDQ 3. INJECT 4. EJECT 5. DISPLAY 6. EXIT

Choice: 3

Enter element to inject: 6

---- DE-QUEUE OPERATIONS ----

1. PUSHDQ 2. POPDQ 3. INJECT 4. EJECT 5. DISPLAY 6. EXIT

Choice: 2

POPDQ 5 SUCCESSFULLY

---- DE-QUEUE OPERATIONS ----

1. PUSHDQ 2. POPDQ 3. INJECT 4. EJECT 5. DISPLAY 6. EXIT

Choice: 5

6

---- DE-QUEUE OPERATIONS ----

1. PUSHDQ 2. POPDQ 3. INJECT 4. EJECT 5. DISPLAY 6. EXIT

Choice: 4

EJECTED 6 SUCCESSFULLY

---- DE-QUEUE OPERATIONS ----

1. PUSHDQ 2. POPDQ 3. INJECT 4. EJECT 5. DISPLAY 6. EXIT

Choice: 6

Output – Polynomial using Array

Polynomial Addition

Polynomial A

Enter Number of Non-Zero terms: 3

(1/3) Enter power: 1

Enter coeffecient: 1

(2/3) Enter power: 2

Enter coeffecient: 2

(3/3) Enter power: 3

Enter coeffecient: 3

Polynomial B

Enter Number of Non-Zero terms: 2

(1/2) Enter power: 0

Enter coeffecient: 0

(2/2) Enter power: 1

Enter coeffecient: 1

A: 3(x ^ 3) + 2(x ^ 2) + 1(x ^ 1) + 0

B: 1(x ^ 1) + 0(x ^ 0) + 0

SUM: 3(x ^ 3) + 2(x ^ 2) + 2(x ^ 1) + 0(x ^ 0) + 0

Output – Standard Searching Algorithms

Enter length of array: 4

Enter elements

arr[0]: 10

arr[1]: 20

arr[2]: 30

arr[3]: 40

\*\*\* Search Algorithms \*\*\*

1. Linear Search

2. Binary Search

3. Exit

Choice: 1

Enter element to search: 20

Linear Search >> 20 found at index 1.

\*\*\* Search Algorithms \*\*\*

1. Linear Search

2. Binary Search

3. Exit

Choice: 2

Enter element to search: 20

Sorted array: 10 20 30 40

Binary Search >> 20 found at index 1.

\*\*\* Search Algorithms \*\*\*

1. Linear Search

2. Binary Search

3. Exit

Choice: 3

Output – Singly Linked List

\*\*\* SINGLY LINKED LIST [SLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

>> 7. Display 8. Exit

Enter choice: 1

Enter data: 1

Successfully inserted 1 at beginning

\*\*\* SINGLY LINKED LIST [SLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

>> 7. Display 8. Exit

Enter choice: 2

Enter data: 2

Successfully inserted 2 at end

\*\*\* SINGLY LINKED LIST [SLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

>> 7. Display 8. Exit

Enter choice: 3

Enter position: 3

Enter data: 3

Successfully inserted 3 at position 3

\*\*\* SINGLY LINKED LIST [SLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

>> 7. Display 8. Exit

Enter choice: 4

Successfully deleted 1 from the beginning

\*\*\* SINGLY LINKED LIST [SLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

>> 7. Display 8. Exit

Enter choice: 7

Cuurent SLL >>> 2 -> 3 -> NULL

\*\*\* SINGLY LINKED LIST [SLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

>> 7. Display 8. Exit

Enter choice: 8

Output – Doubly Linked List

\*\*\* DOUBLY LINKED LIST [DLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

Display >> 7. From Head 8. From Tail

>> 9. Exit

Enter choice: 1

Enter data: 1

Successfully inserted 1

\*\*\* DOUBLY LINKED LIST [DLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

Display >> 7. From Head 8. From Tail

>> 9. Exit

Enter choice: 2

Enter data: 3

Successfully inserted 3

\*\*\* DOUBLY LINKED LIST [DLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

Display >> 7. From Head 8. From Tail

>> 9. Exit

Enter choice: 3

Enter position: 3

Enter data: 3

Successfully inserted 3 at position 3

\*\*\* DOUBLY LINKED LIST [DLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

Display >> 7. From Head 8. From Tail

>> 9. Exit

Enter choice: 5

Successfully deleted 3

\*\*\* DOUBLY LINKED LIST [DLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

Display >> 7. From Head 8. From Tail

>> 9. Exit

Enter choice: 7

Cuurent DLL >>> 1 <-> 3 <-> NULL

\*\*\* DOUBLY LINKED LIST [DLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

Display >> 7. From Head 8. From Tail

>> 9. Exit

Enter choice: 8

Cuurent DLL >>> 3 <-> 1 <-> NULL

\*\*\* DOUBLY LINKED LIST [DLL] \*\*\*

Insert at >> 1. Beginning 2. End 3. Position

Delete at >> 4. Beginning 5. End 6. Position

Display >> 7. From Head 8. From Tail

>> 9. Exit

Enter choice: 9

Output – Polynomial using Linked List

Enter Number of Non-Zero terms: 3

(1/3) Enter power: 0

Enter coeffecient: 0

(2/3) Enter power: 1

Enter coeffecient: 1

(3/3) Enter power: 2

Enter coeffecient: 2

Polynomial A = 2(x ^ 2) + 1(x ^ 1) + 0(x ^ 0) + 0

Enter Number of Non-Zero terms: 2

(1/2) Enter power: 2

Enter coeffecient: 2

(2/2) Enter power: 3

Enter coeffecient: 3

Polynomial B = 3(x ^ 3) + 2(x ^ 2) + 0

SUM = 3(x ^ 3) + 4(x ^ 2) + 1(x ^ 1) + 0(x ^ 0) + 0

Output – Linked Stack

\*\*\* STACK OPTIONS \*\*\*

1. PUSH 2. POP 3. DISPLAY 4. EXIT

Enter choice: 1

Enter data: 5

Successfully Pushed 5

\*\*\* STACK OPTIONS \*\*\*

1. PUSH 2. POP 3. DISPLAY 4. EXIT

Enter choice: 1

Enter data: 6

Successfully Pushed 6

\*\*\* STACK OPTIONS \*\*\*

1. PUSH 2. POP 3. DISPLAY 4. EXIT

Enter choice: 2

Successfully Popped 6

\*\*\* STACK OPTIONS \*\*\*

1. PUSH 2. POP 3. DISPLAY 4. EXIT

Enter choice: 3

Cuurent STACK >>>

5

\*\*\* STACK OPTIONS \*\*\*

1. PUSH 2. POP 3. DISPLAY 4. EXIT

Enter choice: 4

Output – Linked Queue

\*\*\* QUEUE OPTIONS \*\*\*

1. ENQUEUE 2. DEQUEUE 3. DISPLAY 4. EXIT

Enter choice: 1

Enter data: 5

Successfully enqueued 5

\*\*\* QUEUE OPTIONS \*\*\*

1. ENQUEUE 2. DEQUEUE 3. DISPLAY 4. EXIT

Enter choice: 1

Enter data: 6

Successfully enqueued 6

\*\*\* QUEUE OPTIONS \*\*\*

1. ENQUEUE 2. DEQUEUE 3. DISPLAY 4. EXIT

Enter choice: 2

Successfully dequeued 5

\*\*\* QUEUE OPTIONS \*\*\*

1. ENQUEUE 2. DEQUEUE 3. DISPLAY 4. EXIT

Enter choice: 3

Cuurent QUEUE >>> 6

\*\*\* QUEUE OPTIONS \*\*\*

1. ENQUEUE 2. DEQUEUE 3. DISPLAY 4. EXIT

Enter choice: 4

Output – Binary Tree

--- Binary Tree using Array with MAXSIZE 20 ---

### Build Binary Tree ###

Enter Root Item: 5

Does Node [5] have left subtree [1-yes]: 1

>>Enter Item: 4

Does Node [4] have left subtree [1-yes]: 0

Does Node [4] have right subtree [1-yes]: 1

>>Enter Item: 6

Does Node [6] have left subtree [1-yes]: 0

Does Node [6] have right subtree [1-yes]: 0

Does Node [5] have right subtree [1-yes]: 0

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 1

Enter element to search: 5

Element Found.

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 2

\*\*\* Traversal Option \*\*\*

1. Inorder 2. Preorder 3. Postorder

Enter Choice: 1

4 6 5

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 3

Enter Parent Node: 6

Enter item to insert: 7

Child Option: 1. Left 2. Right

Enter Choice : 2

Insertion Successful

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 4

Enter item to delete: 7

Deletion Successful

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 5

Output – Binary Tree using Linked List

--- Binary Tree Using Linked List ---

\*\*\* Build Binary Tree \*\*\*

Enter Root Element: 5

Does Node [5] have left subtree [1-yes]: 1

>>Enter Data: 4

Does Node [4] have left subtree [1-yes]: 0

Does Node [4] have right subtree [1-yes]: 0

Does Node [5] have right subtree [1-yes]: 1

>>Enter Data: 6

Does Node [6] have left subtree [1-yes]: 0

Does Node [6] have right subtree [1-yes]: 0

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 1

Enter element to search: 5

Element Found.

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 2

\*\*\* Traversal Option \*\*\*

1. Inorder 2. Preorder 3. Postorder

Enter Choice: 1

4 5 6

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 3

Enter Parent Node: 6

Enter item to insert: 7

Child Option: 1. Left 2. Right

Enter Choice : 2

Insertion Successful.

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 4

Enter item to delete: 7

Deletion Successful.

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 2

\*\*\* Traversal Option \*\*\*

1. Inorder 2. Preorder 3. Postorder

Enter Choice: 2

5 4 6

\*\*\* Binary Tree Options \*\*\*

1. Search 2. Traverse 3. Insert 4. Delete 5. Exit

Choice: 5

Output – Binary Search Tree

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 1

Enter data to insert: 25

Successfully Inserted.

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 1

Enter data to insert: 35

Successfully Inserted.

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 1

Enter data to insert: 75

Successfully Inserted.

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 2

\*\*\* Traversal Option \*\*\*

1. Inorder(NR) 2. Inorder 3. Preorder 4. Postorder

Enter Choice: 1

25 35 75

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 4

Enter item to delete : 35

Successfully deleted.

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 3

Enter item to search : 35

35 could not be found.

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 2

\*\*\* Traversal Option \*\*\*

1. Inorder(NR) 2. Inorder 3. Preorder 4. Postorder

Enter Choice: 2

25 75

\*\*\* Tree Options \*\*\*

1. Insert 2. Traverse 3. Search 4. Delete 5. Exit

Enter Choice: 5

Output – Graph

\*\*\* GRAPH \*\*\*

Enter number of vertices: 9

Creating Graph with 9 Vertices

L Enter gptr[0]: 1

L Enter gptr[1]: 2

L Enter gptr[2]: 3

L Enter gptr[3]: 4

L Enter gptr[4]: 5

L Enter gptr[5]: 6

L Enter gptr[6]: 7

L Enter gptr[7]: 8

L Enter gptr[8]: 9

Vertices: 1 2 3 4 5 6 7 8 9

No. of Edges for Vertex '1': 4

L Enter Edge 1: 2

L Enter Edge 2: 3

L Enter Edge 3: 7

L Enter Edge 4: 9

No. of Edges for Vertex '2': 3

L Enter Edge 1: 1

L Enter Edge 2: 4

L Enter Edge 3: 8

No. of Edges for Vertex '3': 3

L Enter Edge 1: 1

L Enter Edge 2: 4

L Enter Edge 3: 5

No. of Edges for Vertex '4': 2

L Enter Edge 1: 2

L Enter Edge 2: 3

No. of Edges for Vertex '5': 1

L Enter Edge 1: 3

No. of Edges for Vertex '6': 1

L Enter Edge 1: 7

No. of Edges for Vertex '7': 2

L Enter Edge 1: 1

L Enter Edge 2: 6

No. of Edges for Vertex '8': 2

L Enter Edge 1: 2

L Enter Edge 2: 9

No. of Edges for Vertex '9': 2

L Enter Edge 1: 1

L Enter Edge 2: 8

Generated Graph is:

[1] -> [2] -> [3] -> [7] -> [9] -> NULL

[2] -> [1] -> [4] -> [8] -> NULL

[3] -> [1] -> [4] -> [5] -> NULL

[4] -> [2] -> [3] -> NULL

[5] -> [3] -> NULL

[6] -> [7] -> NULL

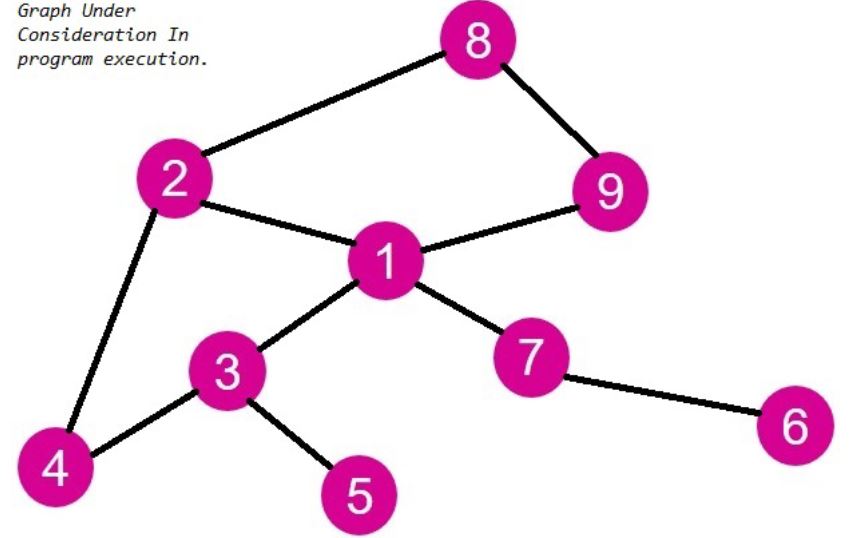
[7] -> [1] -> [6] -> NULL

[8] -> [2] -> [9] -> NULL

[9] -> [1] -> [8] -> NULL

DFS: 1 9 8 2 4 3 5 7 6

BFS: 1 2 3 7 9 4 8 5 6



Output – Standard Sorting Algorithms

Enter Number of Elements: 4

Enter Elements

A[0]: 98

A[1]: 56

A[2]: 80

A[3]: 75

Unsorted array: 98 56 80 75

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 1

swap(98, 56)

Current Array: 56 98 80 75

swap(98, 80)

Current Array: 56 80 98 75

swap(98, 75)

Current Array: 56 80 75 98

Current Array: 56 80 75 98

swap(80, 75)

Current Array: 56 75 80 98

Current Array: 56 75 80 98

Sorted Array: 56 75 80 98

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 6

Enter Number of Elements: 4

Enter Elements

A[0]: 98

A[1]: 56

A[2]: 80

A[3]: 75

Unsorted array: 98 56 80 75

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 2

swap(98, 56)

Current Array: 56 98 80 75

swap(98, 75)

Current Array: 56 75 80 98

Current Array: 56 75 80 98

Sorted Array: 56 75 80 98

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 6

Enter Number of Elements: 4

Enter Elements

A[0]: 98

A[1]: 56

A[2]: 80

A[3]: 75

Unsorted array: 98 56 80 75

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 3

Current Array: 56 98 80 75

Current Array: 56 80 98 75

Current Array: 56 75 80 98

Sorted Array: 56 75 80 98

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 6

Enter Number of Elements: 4

Enter Elements

A[0]: 98

A[1]: 56

A[2]: 80

A[3]: 75

Unsorted array: 98 56 80 75

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 4

Current Array: 56 98 80 75

Current Array: 56 98 75 80

Current Array: 56 75 80 98

Sorted Array: 56 75 80 98

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 6

Enter Number of Elements: 4

Enter Elements

A[0]: 98

A[1]: 56

A[2]: 80

A[3]: 75

Unsorted array: 98 56 80 75

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 5

Pivot = 98

swap(75, 98)

Current Array: 75 56 80 98

Pivot = 75

swap(56, 75)

Current Array: 56 75 80 98

Sorted Array: 56 75 80 98

\*\*\* Sorting Options \*\*\*

1. Bubble 2. Selection 3. Insertion 4. Merge 5. Quick

6. Change array 7. Exit

Enter Choice: 7

Output – Heap Sort

\*\*\* Heap Tree Options \*\*\*

1. Insert 2. Delete 3. Sort 4. Display 5. Exit

Enter Choice: 1

Enter item: 72

Successfully Inserted 72

\*\*\* Heap Tree Options \*\*\*

1. Insert 2. Delete 3. Sort 4. Display 5. Exit

Enter Choice: 1

Enter item: 56

Successfully Inserted 56

\*\*\* Heap Tree Options \*\*\*

1. Insert 2. Delete 3. Sort 4. Display 5. Exit

Enter Choice: 1

Enter item: 98

Successfully Inserted 98

\*\*\* Heap Tree Options \*\*\*

1. Insert 2. Delete 3. Sort 4. Display 5. Exit

Enter Choice: 3

N = 3

Sorting Tree: 98 56 72

\*First Swap 98 <-> 72

Sorting Tree: 72 56 98

\*First Swap 72 <-> 56

Sorted Tree: 56 72 98