Report on Problem 5:

Files included in assignment:

Source Code, Report.

Datastructres used:

Array.

Program outputs:

1. Add operation:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Abdul\Desktop\programs> cd "c:\Users\Abdul\Desktop\programs\assignment 5\" ; if ($?) { gcc array.c -o array } ; if ($?) { .\array }

Original array: 10 , 20 , 30 , 40

_____________________________________________________________

Enter 1 to  Add an element to the array.
Enter 2 to  Insert an element at the desired location in an array.
Enter 3 to  Delete an element from an array.
Enter 4 to  Update an element/information at an index of the array.
Enter 5 to  Find max and min in array.

_____________________________________________________________
1


You pressed 1.

Enter the element to insert in the array.
50

You entered 50.

Added successfully

New Array: 10 20 30 40 50

Space used= 6 units,      i.e. size of input array + 2         NOTE: 1 unit= sizeof(int) bytes        // it is machine dependent
Statements executed= 14,  i.e. (3 * size of input array) + 2

PS C:\Users\Abdul\Desktop\programs\assignment 5>
```

Note:

Calculations on space and time complexity are integrated in source code. Further discussion in algorithm section.

2. Insert at a desired location:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Abdul\Desktop\programs> cd "c:\Users\Abdul\Desktop\programs\assignment 5\" ; if ($?) { gcc array.c -o array } ; if ($?) { .\array }

Original array: 10 , 20 , 30 , 40

_____________________________________________________________

Enter 1 to  Add an element to the array.
Enter 2 to  Insert an element at the desired location in an array.
Enter 3 to  Delete an element from an array.
Enter 4 to  Update an element/information at an index of the array.
Enter 5 to  Find max and min in array.

_____________________________________________________________
2


You pressed 2.

Enter the element to insert in the array.
25

You entered 25.

Enter the index( >=0 && < 4) in the array.
2

You entered 2.

Inserted successfully

New Array: 10 20 25 30 40

Space used= 7 units,      i.e. size of input array + 3         NOTE: 1 unit= sizeof(int) bytes           // it is machine dependent
Statements executed= 16,  i.e. (3 * size of input array) + 4

PS C:\Users\Abdul\Desktop\programs\assignment 5>
```

3. Delete an element from an array:

```
PS C:\Users\Abdul\Desktop\programs\assignment 5> cd "c:\Users\Abdul\Desktop\programs\assignment 5\" ; if ($?) { gcc array.c -o array } ; if ($?) { .\array }

Original array: 10 , 20 , 30 , 40

_____________________________________________________________

Enter 1 to  Add an element to the array.
Enter 2 to  Insert an element at the desired location in an array.
Enter 3 to  Delete an element from an array.
Enter 4 to  Update an element/information at an index of the array.
Enter 5 to  Find max and min in array.

_____________________________________________________________
3


You pressed 3.

Enter the element to delete in the array.
30

You entered 30.

Deleted successfully

New Array: 10 20 40

Space used= 4 units,      i.e. size of input array       NOTE: 1 unit= sizeof(int) bytes           // it is machine dependent
Statements executed= 13,  i.e. 3 * size of input array

PS C:\Users\Abdul\Desktop\programs\assignment 5>
```

4. Update an element at an index of the array:

```
PS C:\Users\Abdul\Desktop\programs\assignment 5> cd "c:\Users\Abdul\Desktop\programs\assignment 5\" ; if ($?) { gcc array.c -o array } ; if ($?) { .\array }

Original array: 10 , 20 , 30 , 40


_____________________________________________________________________

Enter 1 to  Add an element to the array.
Enter 2 to  Insert an element at the desired location in an array.
Enter 3 to  Delete an element from an array.
Enter 4 to  Update an element/information at an index of the array.
Enter 5 to  Find max and min in array.

_____________________________________________________________________
4


You pressed 4.

Enter the index( >=0 && < 4) in the array to update.
2

You entered 2.

Enter the element.
27

You entered 27.

Updated successfully

New Array: 10 20 27 40

Space used= 0 units,         NOTE: 1 unit= sizeof(int) bytes          // it is machine dependent
Statements executed= 2

PS C:\Users\Abdul\Desktop\programs\assignment 5>
```

5. Find maxium and minium element in the array:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Abdul\Desktop\programs> cd "c:\Users\Abdul\Desktop\programs\assignment 5\" ; if ($?) { gcc array.c -o array } ; if ($?) { .\array }

Original array: 10 , 20 , 30 , 40


_____________________________________________________________

Enter 1 to  Add an element to the array.
Enter 2 to  Insert an element at the desired location in an array.
Enter 3 to  Delete an element from an array.
Enter 4 to  Update an element/information at an index of the array.
Enter 5 to  Find max and min in array.

_____________________________________________________________
5


You pressed 5.

Maxium element: 40
Minium element: 10

Space used= 2 units        Note: 1 unit=sizeof(int)        //machine dependent
Statements executed= 13,  // may vary, dependent on input array

PS C:\Users\Abdul\Desktop\programs\assignment 5>
```

Algorithms used:

1.Algorithm for add method:

Assuming, size of input array= N, and adding at the end.

step 1: declare an array (say new_array) of size= N + 1

space used= N+1, cpu steps= 0

step 2: copy all elements of the original array into new_array

space used= 1 (for index keeping variable (like " i " in for loop)) ,

cpu steps= 1(initializing i) + N (assignments) + N (conditions checks) + N (iterations)

step 3; assign the element (to be added) at the last index of new_array.

space used= 0, cpu steps= 1(assignment)

Hence, For add method:

space complexity= (N+1) + (1) + (0) = N+2 = O(N)

time complexity= (0) + (3N + 1) + (1) = 3N+2 = O(N)

2.Algorithm for insertion at a desired place:

Suppose N is the number of elements in the original array.

We take two inputs from user: index(at which insertion takes place). Assume it is " x ".

and element to be inserted. Lets assume it is " t ".

In this algorithm we (logically) divide the original array into two parts.

part 1 is the array before index " x "

part 2 is the array after index " x "

eg. suppose we are inserting at index "x= 2(say)"

array= {1,2,3,4,5}

part1= {1,2} //indexes 0 and 1 are possible, i.e. 0 to (x-1)

part 2= {4,5} //index 3 and 4 are possible, in general indexes from (x+1) to (N-1)

part 1 and part 2 are then copied into newly created array in two " for loops ".

step 1; check if x is valid

space used= 0, cpu steps= 1 (condition check)

step 2; declare an array(say new_array) of size= N + 1

space used= N+1, cpu steps= 0

step 3; copy elements of the original array into new_array before x, i.e. from indexes 0 to (x-1)

space used= 1 (for index keeping variable (like "i" in for loop)) ,

cpu steps= 1 (initialize i) + x (assignments) + x (conditions checks) + x (iterations)

step 4; assign the element(t) at x'th index in the new_array.

space used= 0, cpu steps= 1(assignment)

step 5; copy elements of the original array into new_array after x, i.e. from (x+1) to (N-1) index

space used= 1 (for index keeping variable (like "i" in for loop)) ,

cpu steps= 1 (initialize i) + N - x (assignments) + N - x (conditions checks) + N - x (iterations)

Hence, For insert at a particular position method:

space complexity= (0) + (N+1) + (1) + (0) + (1) = N+3 = O(N)

time complexity= (1) + (0) + (3x + 1) + (1) + (3(N-x) + 1) = 3N+4 = O(N)

3.Algorithm to delete

Assuming that user is deleting element "t" at index "x" somewhere in array which has N total elements.

step 1; declare an array(say new_array) of size = N - 1

space used= N-1, cpu steps= 0

step 2; copy elements of the original array into new_array before x, or before t is found

space used= 1 (for index keeping variable (like "i" in for loop))

cpu steps= 1 (initialize i) + (x-1) (assignments) + (x-1) (conditions checks) + (x-1) (iterations)

step 3; if (t is not found after transversing all array)

output "not found" and exit

space used= 0, cpu steps= 1(condition check)

else

increment the index and proceed to copy from (x+1)th to (N-1)th element

space used= 0,

cpu steps= 1(exit condition check) + 1(increment i) + N-x (condition check) + N-x (assignments) +

N-x (iterations)

Hence, For delete method:

space complexity= (N-1) + (1) + (0) + (0) = N = O(N)

time complexity= (0) + (1+3(x-1)) + (1+1+3*(N-x)) = 3N = O(N)

4. Algorithm to update

Assuming that user is updating at "x"th index in array which has N elements in total.

step 1; check if x is valid

space used= 0, cpu steps= 1

step 2; assign new value at x

space used= 0, cpu steps= 1(assignment)

Hence, For update method:

space complexity= (0) + (0) = 0 = O(1)

time complexity= (1) + (1) = 2 = O(1)

5.Algorithm to find minimum and maximum value

step 1; declare and initialize two variables, min = [more than any value in the given array] and max=0, to keep
* track of the minimum and maximum values seen so far while transversal.

space used= 2, cpu steps= 2(assignments)

step 2; transverse the array using for loop

if (current element is greater than max)

update max variable.

if (current element is smaller than min)

update min variable.

space and time calculations for this step:

case 1: assuming trivial case, i.e. array in descending or ascending order

space used= 0

cpu steps= 2(condition checks) + N (assignments each of the variables is being updated in all iterations)

case 2: assuming not ordered.

space used= 0

cpu steps= between { 2 (condition checks and no update at all iterations) and

2 + N (trivial case) }

Hence, For finding min and max function,

space complexity= (2) + (0) = 2 = O(1)

time complexity= (2) + (2+N) = N+4 = O(N)