|  |
| --- |
| **// C program for implementation of selection sort**#include <stdio.h> void swap(int \*xp, int \*yp){    int temp = \*xp;    \*xp = \*yp;    \*yp = temp;} void selectionSort(int arr[], int n){    int i, j, min\_idx;     // One by one move boundary of unsorted subarray    for (i = 0; i < n-1; i++)    {        // Find the minimum element in unsorted array        min\_idx = i;        for (j = i+1; j < n; j++)          if (arr[j] < arr[min\_idx])            min\_idx = j;         // Swap the found minimum element with the first element        swap(&arr[min\_idx], &arr[i]);    }} /\* Function to print an array \*/void printArray(int arr[], int size){    int i;    for (i=0; i < size; i++)        printf("%d ", arr[i]);    printf("\n");} // Driver program to test above functionsint main(){    int arr[] = {64, 25, 12, 22, 11};    int n = sizeof(arr)/sizeof(arr[0]);    selectionSort(arr, n);    printf("Sorted array: \n");    printArray(arr, n);    return 0;} |
|  |

**Data Structures & Algorithms**

 **Solved Sample Assignments**

Output:

Sorted array:

11, 12, 22, 25, 64

|  |
| --- |
| **// C program for implementation of Bubble sort**#include <stdio.h> void swap(int \*xp, int \*yp){    int temp = \*xp;    \*xp = \*yp;    \*yp = temp;} **// A function to implement bubble sort**void bubbleSort(int arr[], int n){   int i, j;   for (i = 0; i < n-1; i++)              // Last i elements are already in place          for (j = 0; j < n-i-1; j++)            if (arr[j] > arr[j+1])              swap(&arr[j], &arr[j+1]);} /\* Function to print an array \*/void printArray(int arr[], int size){    int i;    for (i=0; i < size; i++)        printf("%d ", arr[i]);    printf("n");} // Driver program to test above functionsint main(){    int arr[] = {64, 34, 25, 12, 22, 11, 90};    int n = sizeof(arr)/sizeof(arr[0]);    bubbleSort(arr, n);    printf("Sorted array: \n");    printArray(arr, n);    return 0;} |

Output:

Sorted array:

11, 12, 22, 25, 34, 64, 90

|  |
| --- |
| // C program for insertion sort#include <stdio.h>#include <math.h> /\* Function to sort an array using insertion sort\*/void insertionSort(int arr[], int n){   int i, key, j;   for (i = 1; i < n; i++)   {       key = arr[i];       j = i-1;        /\* Move elements of arr[0..i-1], that are          greater than key, to one position ahead          of their current position \*/       while (j >= 0 && arr[j] > key)       {           arr[j+1] = arr[j];           j = j-1;       }       arr[j+1] = key;   }} // A utility function to print an array of size nvoid printArray(int arr[], int n){   int i;   for (i=0; i < n; i++)       printf("%d ", arr[i]);   printf("\n");}   /\* Driver program to test insertion sort \*/int main(){    int arr[] = {12, 11, 13, 5, 6};    int n = sizeof(arr)/sizeof(arr[0]);     insertionSort(arr, n);    printArray(arr, n);     return 0;} |

Output:

5, 6, 11, 12, 13

Linear Search

/\* Linearly search x in arr[].  If x is present then return it location,  otherwise return -1\*/

int search(int arr[], int n, int x)

{

    int i;

    for (i=0; i<n; i++)

        if (arr[i] == x)

         return i;

    return -1;

}

**Stacks**

|  |
| --- |
| #include<iostream>#include<stdlib.h> using namespace std; class twoStacks{    int \*arr;    int size;    int top1, top2;public:   twoStacks(int n)  // constructor   {       size = n;       arr = new int[n];       top1 = -1;       top2 = size;   }    // Method to push an element x to stack1   void push1(int x)   {       // There is at least one empty space for new element       if (top1 < top2 - 1)       {           top1++;           arr[top1] = x;       }       else       {           cout << "Stack Overflow";           exit(1);       }   }    // Method to push an element x to stack2   void push2(int x)   {       // There is at least one empty space for new element       if (top1 < top2 - 1)       {           top2--;           arr[top2] = x;       }       else       {           cout << "Stack Overflow";           exit(1);       }   }    // Method to pop an element from first stack   int pop1()   {       if (top1 >= 0 )       {          int x = arr[top1];          top1--;          return x;       }       else       {           cout << "Stack UnderFlow";           exit(1);       }   }    // Method to pop an element from second stack   int pop2()   {       if (top2 < size)       {          int x = arr[top2];          top2++;          return x;       }       else       {           cout << "Stack UnderFlow";           exit(1);       }   }};  /\* Driver program to test twStacks class \*/int main(){    twoStacks ts(5);    ts.push1(5);    ts.push2(10);    ts.push2(15);    ts.push1(11);    ts.push2(7);    cout << "Popped element from stack1 is " << ts.pop1();    ts.push2(40);    cout << "\nPopped element from stack2 is " << ts.pop2();    return 0;} |

Output:

 Popped element from stack1 is 11

 Popped element from stack2 is 40