Introduction to programming Dr. Saleh Mosbah

Name: Abdallah Mohamed Elsayed Aboudeif Awad

Registration No. 19235004

**Final exam report**

**Searching and sorting algorithms**

**SEARCHING AND SORTING**

Sorting and Searching are fundamental operations in computer science. Sorting refers to the operation of arranging data in some given order. Searching refers to the operation of searching the particular record from the existing information.

**Searching**

Searching refers to the operation of finding the location of a given item in a collection of items structure.

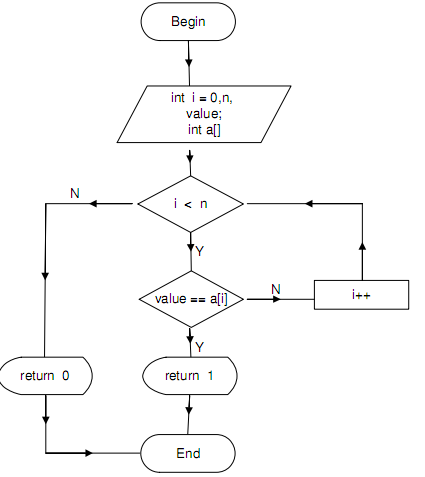
The search is said to be successful if ITEM does appear in DATA and unsuccessful otherwise.

Any algorithm, which performs a search operation, accepts an argument we call as "ele" and tries to find an occurrence of the key "ele" in some structure. It is possible that the search for a particular element in a structure is unsuccessful then the algorithm should return a special message to indicate an unsuccessful search.

A number of methods are available to perform this operation, two important methods here are Linear Search and Binary search.

**LINEAR SEARCH**

The most intuitive way to search for a given ITEM in Data Structure is to compare ITEM with each element of Data structure one by one. The search starts by sequentially comparing the elements of the array one after the other from the beginning to the end with the element to be searched.



|  |
| --- |
| Algorithm: A is an array of **N** elements and **ele** is the element being searched in the array. |
| Step 1: LOC = -1  Step 2: For I= 0 TO N-1 Do  Step 3: If(ele = A[I] Then  Step 4: LOC = I  Step 5: Goto Step 6  [EndIf]  [End of For loop]  Step 6: If(LOC >= 0) Then  Step 7: Print ele, “Found in location”, LOC  Step 8: Else  Print ele, “Not found”  [EndIf]  Step 10: Exit |

|  |
| --- |
| **Program 1:**Program to perform linear search in C language. |
| #include <stdio.h>  #include <conio.h>  int linear\_search (int\*, int, int);  void main()  {  int A[100],ele,pos,n,i;  clrscr();  printf(“Enter the size of the array:”);  scanf(“%d”, &n);  printf(“\n Enter %d elements\n”, n);  for(i=0; i<n; i++)  scanf(“%d”, &A[i]);  printf(“\n Enter element to search:”);  scanf(“%d”, &ele);  pos = Linear\_search(A,ele,n);  if(pos==-1)  printf(“\n Element not present”);  else  printf(“\n Element found in position: %d”, pos);  }  **/\* Function to search an item using linear search technique \*/**  int linear\_search(int a[ ], int e, int m)  {  int i;  for(i=0; i<m; i++)  if(a[i]==e) return (i+1);  return(-1);  } |

|  |
| --- |
| **Program 2:**Program to perform linear search in Python language. |
| items = ["Mohamed", "Ahmed", "Ali","Gaber","Fathy","Omar","Sayed","Hassan","Raouf"]  print ("List items are: " , items )  print ("please type the iteam you want to search for: ")  ele=input ()  # You can switch between 2 methods of linear search, change the switch value to select method 1 or 2  switch =2  if switch ==1:  loc = -1  for i in range (len(items)-1):  if (ele == items[i]):  loc=i  if (loc>=0):  print(ele,"found in location",loc)  else:  print(ele,"not found")    if switch==2:  if ele in items:  loc = items.index(ele)  print(ele,"is found in location",loc)  else:  print("Sorry,",ele,"is not found!")  input("Program has completed") |

**BINARY SEARCH**

Binary search is considered the most efficient method of searching a linear array without the use of auxiliary indices. The element to be compared is compared with the element of the middle element of the array. If the elements are the same, the search ends successfully; otherwise, the search should continue either to the left or to the right of the array in a similar manner. It may be noted that the binary search can be applied if and only if the array is sorted in some form of order.

The steps of binary search may be summarized as follows:

1. Find the position of the middle element of the array.

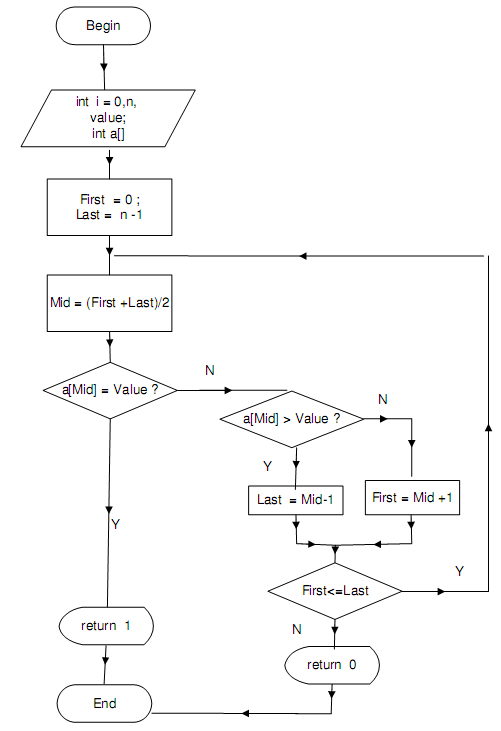
2. Compare the element in the middle position with the search element.

3. One of the following actions may be performed after the comparison.

a. If the search element is the same as the middle element then note its position.

b. Otherwise if the search element is less than the element in the middle position then continue the search to the left portion of the middle element.

c. Otherwise if the search element is greater than the element in the middle position then continue the search to the right portion of the mid.



|  |
| --- |
| **Algorithm 5.2:Binary\_search (A, ele, N) –A**is a sorted array of **N** elements and **ele** is the element being searched in the array. **LOW** and **HIGH** identify the positions of the first and last elements in a range and **MID** identifies the position of the middle element. |
| Step 1: LOW = 0  Step 2: HIGH = N-1  Step 3: LOC = -1  Step 4: While (LOW <= HIGH) Do  Step 5: MID = (LOW+HIGH)/2  Step 6: If(ele = A[MID]) Then  LOC = MID  GO TO Step 9  [EndIf]  Step 7: If(ele < A[MID]) Then  HIGH = MID-1  Step 8: Else  LOW = MID+1  [EndIf]  [End of while loop]  Step 9: If(LOC >= 0) Then  Print ele, Foundin location”, LOC  Step 10: Else  Step 11: Print ele, “Not found”  [EndIf]  Step 12: Exit |

|  |
| --- |
| **Program 1:**Program to perform binary search in C language |
| #include <stdio.h>  #include <conio.h>  int Binary\_search (int\*, int, int);  int main()  {  int A[100],ele,loc,n,i;  //system("cls");  printf("Enter the size array size:");  scanf("%d", &n);  printf("\n Enter %d elements in sorted order\n", n);  for(i=0; i<n; i++)  scanf("%d", &A[i]);  printf("\n Enter the element to search:");  scanf("%d", &ele);  loc= Binary\_search(A,ele,n);  if(loc==-1)  printf(" Element not present\n");  else  printf("Element found in %d position\n", loc+1);  }  /\* Function to perform Binary search \*/  int Binary\_search(int A[ ], int e, int m)  {  int low,high, mid;  low = 0;  high = m-1;  while (low <= high)  {  mid = (low + high)/2;  if(e == A[mid]) {  return (mid) ;  if(e < A[mid]) ;  high = mid-1;  }else low = mid+1;  }  return(-1);  } |

|  |
| --- |
| **Program 2:**Program to perform binary search in Python language |
| # Returns index of ele in arr if present, else -1  def binary\_search(arr, low, high, ele):  # Check base case  if high >= low:  mid = (high + low) // 2  # If element is present at the middle itself  if arr[mid] == ele:  return mid  # If element is smaller than mid, then it can only  # be present in left subarray  elif arr[mid] > ele:  return binary\_search(arr, low, mid - 1, ele)  # Else the element can only be present in right subarray  else:  return binary\_search(arr, mid + 1, high, ele)  else:  # Element is not present in the array  return -1  # Input searchkey  arr = [ 2, 3, 4, 6, 7,12,25,28,29,33,34,38,41,54,57 ]  print ("List items are: " , arr)  ele=int(input("please type the iteam you want to search for: "))  # Function call  result = binary\_search(arr, 0, len(arr)-1, ele)  if result != -1:  print(ele,"is found in location [", str(result),"]")  else:  print("Sorry,",ele,"is not found!")  input("Program has completed") |

**SORTING**

Sorting refers to arranging of data elements in some type of order. Ordering or sorting of data with some relationship is of fundamental importance. Certain factors however should be considered before designing a sort algorithm.

There is a number of sorting techniques:

1. Bubble sort

2. Selection sort

3. Insertion sort

4. Merge sort

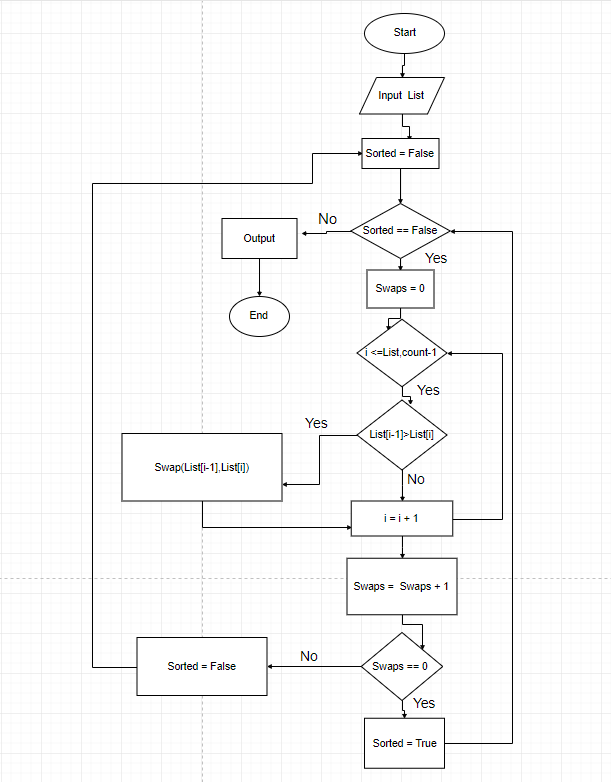
5. Heap sort

6. Quick sort

**BUBBLE SORT**

In this sorting algorithm, multiple swapping take place in one iteration. Smaller elements move or ‘bubble’ up to the top of the list. In this method, we compare the adjacent members of the list to be sorted, if the item on top is greater than the item immediately below it, they are swapped.

During the next pass the same steps are repeated from the beginning of the array, however this time the comparisons are only for n-l elements. The process is repeated again and again until only two elements are left for comparison. The last iteration ensures that the first two elements of the array are placed in the correct order.



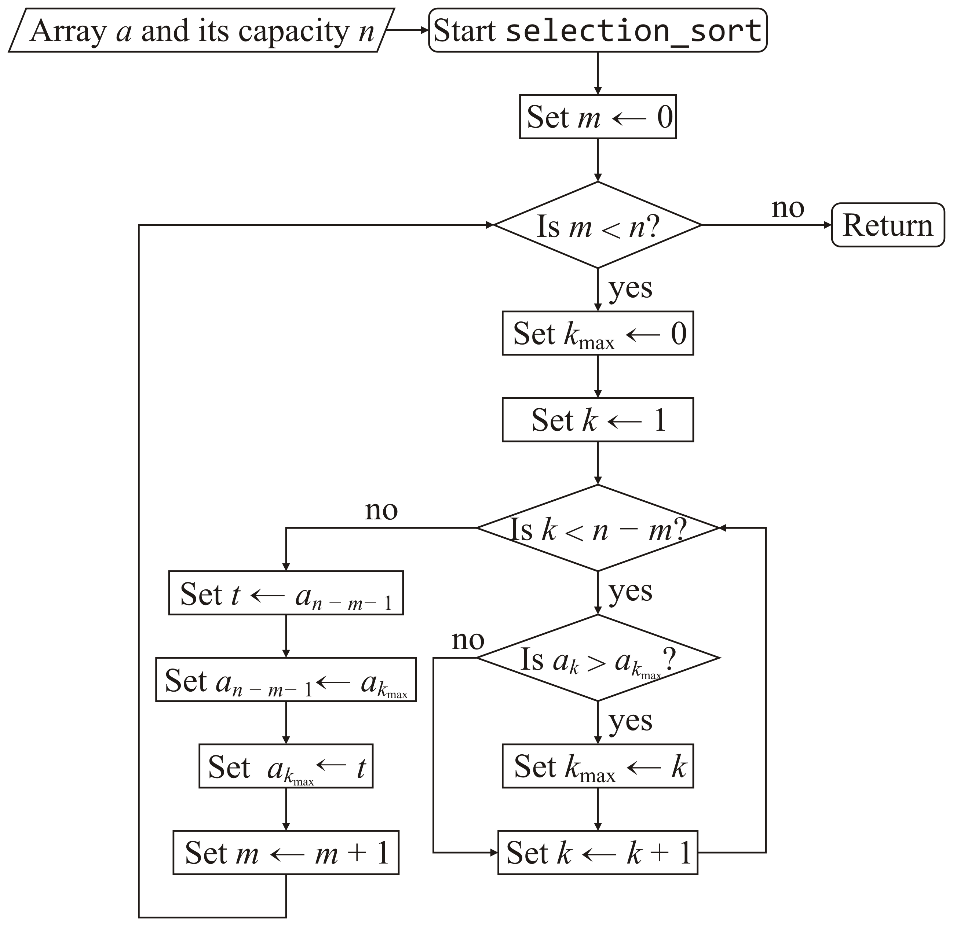
|  |
| --- |
| **Algorithm 5.3: Bubble\_sort (A,N)**Given an array **A** of **N** elements, this procedure sorts the elements in the ascending order using the method described above. The variables **I** and **J** are used to index the array elements. |
| Step 1: For I = 1 to N –1 Do  Step 2: For J = 0 to N –2Do  Step 3: [Compare adjacent elements]  If (A[J] > A[J + 1]) Then  [Exchange the values]  Step 4: Temp = A[J]  A[J] = A[J + 1]  A[J + 1] = Temp  Step 5: [End If]  [End of Step 2 For loop]  [End of Step 1 For loop]  Step 6: Exit |

|  |
| --- |
| **Program 1:** Program to arrange **N** numbers in the ascending and descending order using bubble sort method in C language. |
| #include<stdio.h>  #include<conio.h>  void main()  {  int i,j,n,A[100],temp;  clrscr();  printf("Enter number of elements:");  scanf("%d",&n);  printf("Enter those element:\n");  for(i=0;i<n;i++)  {  scanf("%d",&A[i]);  }  for(i=0;i<n;i++)  {  for(j=0;j<n-1;j++)  {  if(A[j]>A[j+1])  {  temp=A[j];  A[j]=A[j+1];  A[j+1]=temp;  }  }  }  printf("After sorting here is the elements in ascending order:\n");  for(i=0;i<n;i++)  {  printf("%d\t",A[i]);  }  getch();  } |

|  |
| --- |
| Program 2 for implementation of Bubble Sort in Python |
| def bubbleSort(arr):  n = len(arr)  # Traverse through all array elements  for i in range(n):  # Last i elements are already in place  for j in range(0, n-i-1):  # traverse the array from 0 to n-i-1  # Swap if the element found is greater  # than the next element  if arr[j] > arr[j+1] :  arr[j], arr[j+1] = arr[j+1], arr[j]  # Sorting arr array  arr = [64, 34, 25, 12, 22, 11, 90]  print ("List items are: " , arr)  bubbleSort(arr)  print ("Sorted array is:")  for i in range(len(arr)):  print ("[",i,"]",arr[i])  input("Program has completed") |

**SELECTION SORT**

In this sorting we find the smallest element in this list and put it in the first position. Then find the second smallest element in the list and put it in the second position. And so on.



|  |
| --- |
| **Algorithm 5.4: Selection\_sort (A,N)**–Given an array **A** of **N** elements, this procedure sorts the elements in the ascending order using the method described above. The variables **I** and **J** are used to index the array elements in C language |
| Step 1: For I = 0 TO N –2 Do  Step 2: [Assume ith elements as smallest]  Small = A[I]  Step 3: POS = I  Step 4: [Find the smallest element in the array and its position]  For J = I + 1 TO N –1 Do  Step 5: If(A[J] < small) Then  Step 6: small = A[J]  Step 7: POS = J  [EndIf]  [End of Step 4 For loop]  Step 8: [Exchange ith element with smallest element]  A[POS] = A[I]  Step 9: A[I] = small  [End of Step 1 For loop]  Step 10: Exit |

|  |
| --- |
| Program 1: for implementation of selection sort in C language |
| #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 functions  int 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;  } |

|  |
| --- |
| Program 1: for implementation of selection sort in Python language |
| def selectSort(arr):  n = len(arr)  # Traverse through all array elements  for i in range(n):    # Find the minimum element in remaining  # unsorted array  min\_idx = i  for j in range(i+1, n):  if arr[min\_idx] > arr[j]:  min\_idx = j    # Swap the found minimum element with  # the first element  arr[i], arr[min\_idx] = arr[min\_idx], arr[i]  print ("Sorted array is:")  for i in range(n):  print ("[",i,"]",arr[i])  # Sorting arr array  items = [64, 34, 25, 12, 22, 11, 90,0]  print ("List items are: " , items)  selectSort(items)  input("Program has completed") |