DATA STRUCTURES AND ALGORITHMS

Arrays

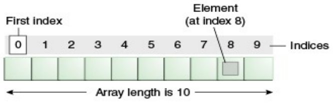
By
Zainab Malik

Content

- Concept of Arrays
- Array Representation
- Operations performed on arrays
- Limitation of arrays
- Application of linear Arrays

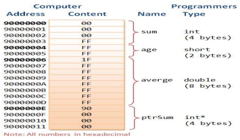
Arrays

- A linear array is a list of finite number of homogeneous data elements such that
  - The elements of the array are referenced by an index set consisting of n consecutive integer numbers
  - The elements of the array are stored in consecutive memory locations



Array Representation in Memory

- Memory of a computer system is simply a sequence of addressed location.



- As arrays are stored in consecutive memory locations, the system need not to keep track of the address of every element of that array, but needs to keep the address of first element only (base address)

Array Representation in Memory

Address	Slot
1197	15
1198	
1199	14
1200	
1201	23
1202	
1203	11
1204	
1205	42
1206	
1207	
1208	
1209	

0	1	2	3	4
15	14	23	11	42

array

Loc (array[k])= base(array) + w* k
w is # of bytes per element
K is index Number

Loc(array[3])= 1197+2*3=1203

Operations performed on arrays

- Traversal Operation
- Searching Operation
  - Linear Search
  - Binary Search
- Insertion Operation
- Deletion Operation
- Sorting Operation
  - Selection Sort
  - Bubble Sort
  - Insertion Sort
  - Merge Sort
  - Quick Sort etc.

Traversal Operation

TraverseLinearArray(a, n)

*Here **a** is a linear array of size **n**. This algorithm traverses the array and applies certain operation to each element of the array.*

```
1. Set i=0                                  //Initialize counter
2. Repeat steps 3 and 4 while i<= (n-1)
3.     Apply process (a[i])                 //visit element
4.     Set i=i+1                            //increment counter
5. Endwhile
6. Exit
```

0	1	2	3	4	5
15	14	23	11	42	03

Searching

Searching Operation: Linear Search

LinearSearch(a, n, item, loc)

*Here **a** is a linear array of size **n**. This algorithm finds the location of the **item** in linear array **a**. If search end in success it sets **loc** to the index of the element; otherwise it sets **loc** to -1.*

```
1. Set i=0                                  //Initialize counter
2. Repeat steps 3 and 4 while i<= (n-1)
3.     if(a[i]=item) then
           Set loc=i                        //item found at location I
           Exit
       Endif
4.     Set i=i+1                            //increment counter
5. Endwhile
6. Set loc=-1                               //item not found
7. Exit
```

0	1	2	3	4	5	6
3	10	15	20	35	40	60

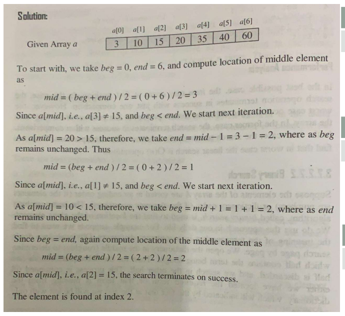
Searching Operation: Binary Search

BinarySearch(a, n, item, loc)

*Here **a** is a linear array of size **n**. This algorithm finds the location of the **item** in sorted (asdending) linear array **a**. If search end in success it sets **loc** to the index of the element; otherwise it sets **loc** to -1. here variables **beg** and **end** are used to keep track of the first and last element of the array and variable **mid** is used as index of the middle element of the array under consideration.*

```
1. Set beg=0
2. Set end=n-1
3. Set mid= (beg+end)/2
4. Repeat steps 5 and 6 while ((beg ≤ end) && (a[mid] ≠ item))
5.     if(item < a[mid]) then
           Set end=mid-1
       else
           Set beg=mid+1
       Endif
6.     Set mid= (beg+end) /2                // shift mid
7. Endwhile
8. If( beg > end ) then
          Set loc=-1                        //item not found
     Else
          Set loc =mid                      //item found at mid
     Endif
9. Exit
```

Searching Operation: Binary Search



Solution:

Given Array *a*

a[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]	*a*[5]	*a*[6]
3	10	15	20	35	40	60

To start with, we take *beg* = 0, *end* = 6, and compute location of middle element as

$$mid = (beg + end) / 2 = (0 + 6) / 2 = 3$$

Since *a*[*mid*], *i.e.*, *a*[3] ≠ 15, and *beg* < *end*. We start next iteration.

As *a*[*mid*] = 20 > 15, therefore, we take *end* = *mid* − 1 = 3 − 1 = 2, where as *beg* remains unchanged. Thus

$$mid = (beg + end) / 2 = (0 + 2) / 2 = 1$$

Since *a*[*mid*], *i.e.*, *a*[1] ≠ 15, and *beg* < *end*. We start next iteration.

As *a*[*mid*] = 10 < 15, therefore, we take *beg* = *mid* + 1 = 1 + 1 = 2, where as *end* remains unchanged.

Since *beg* = *end,* again compute location of the middle element as

$$mid = (beg + end) / 2 = (2 + 2) / 2 = 2$$

Since *a*[*mid*], *i.e.*, *a*[2] = 15, the search terminates on success.

The element is found at index 2.

0	1	2	3	4	5	6
3	10	15	20	35	40	60

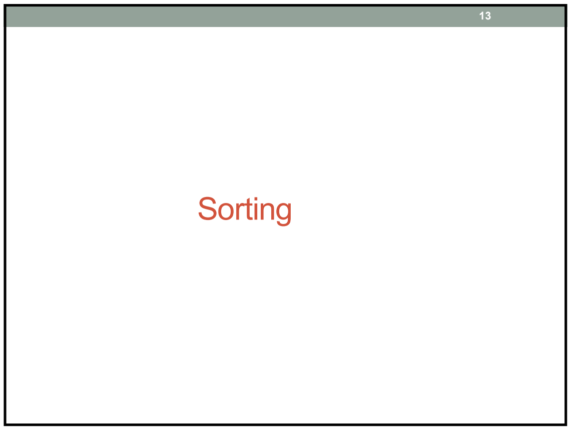
0	1	2	~~3~~	4	~~5~~	~~6~~
3	10	15	~~20~~	~~35~~	~~40~~	~~60~~

0	1	2	~~3~~	4	~~5~~	~~6~~
3	10	15	~~20~~	~~35~~	~~40~~	~~60~~

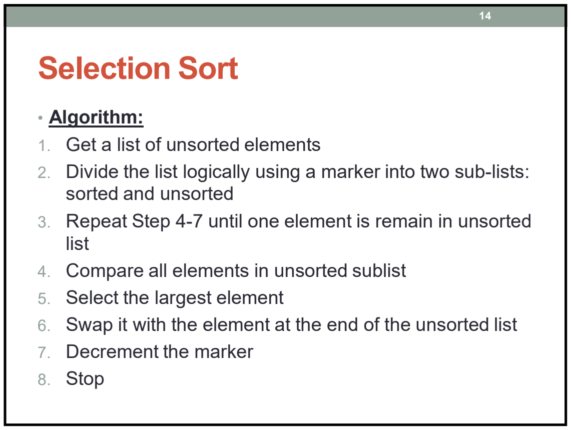
Class Task

Write a function to find the location of largest element in an array.

Signature of function must as follow
Int Largest(int arr[], int n)



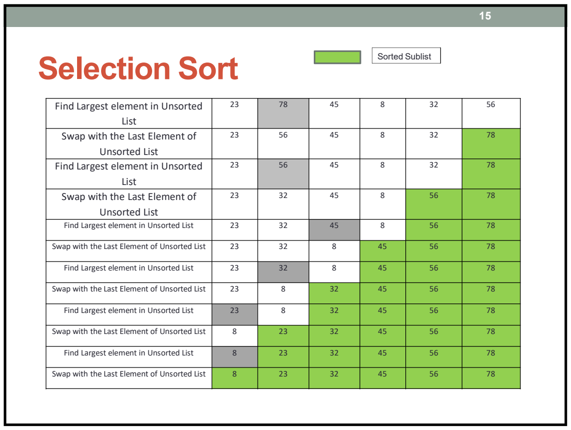
Sorting



Selection Sort

- **Algorithm:**

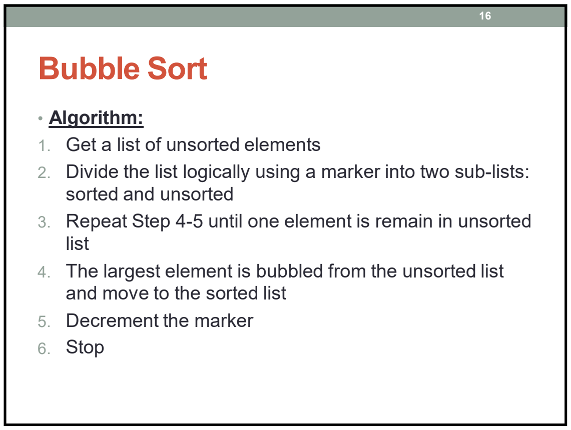
1. Get a list of unsorted elements
2. Divide the list logically using a marker into two sub-lists: sorted and unsorted
3. Repeat Step 4-7 until one element is remain in unsorted list
4. Compare all elements in unsorted sublist
5. Select the largest element
6. Swap it with the element at the end of the unsorted list
7. Decrement the marker
8. Stop



Selection Sort

🟩 Sorted Sublist

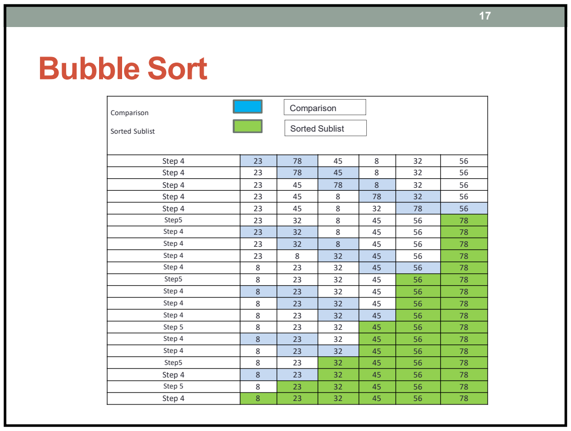
Find Largest element in Unsorted List	23	78	45	8	32	56
Swap with the Last Element of Unsorted List	23	56	45	8	32	78
Find Largest element in Unsorted List	23	56	45	8	32	78
Swap with the Last Element of Unsorted List	23	32	45	8	56	78
Find Largest element in Unsorted List	23	32	45	8	56	78
Swap with the Last Element of Unsorted List	23	32	8	45	56	78
Find Largest element in Unsorted List	23	32	8	45	56	78
Swap with the Last Element of Unsorted List	23	8	32	45	56	78
Find Largest element in Unsorted List	23	8	32	45	56	78
Swap with the Last Element of Unsorted List	8	23	32	45	56	78
Find Largest element in Unsorted List	8	23	32	45	56	78
Swap with the Last Element of Unsorted List	8	23	32	45	56	78



Bubble Sort

- **Algorithm:**

1. Get a list of unsorted elements
2. Divide the list logically using a marker into two sub-lists: sorted and unsorted
3. Repeat Step 4-5 until one element is remain in unsorted list
4. The largest element is bubbled from the unsorted list and move to the sorted list
5. Decrement the marker
6. Stop

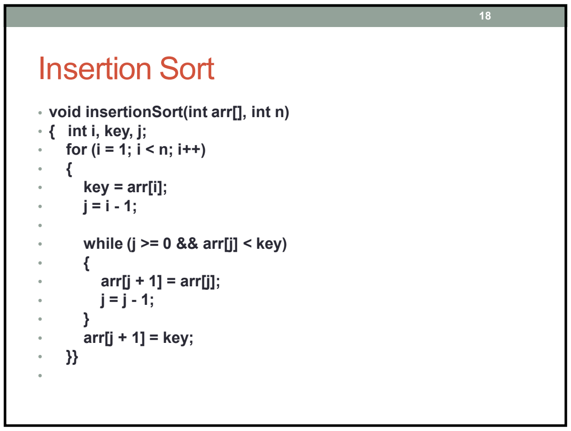


Bubble Sort

Comparison Sorted Sublist						
Step 4	23	78	45	8	32	56
Step 4	23	78	45	8	32	56
Step 4	23	45	78	8	32	56
Step 4	23	45	8	78	32	56
Step 4	23	45	8	32	78	56
Step5	23	32	8	45	56	78
Step 4	23	32	8	45	56	78
Step 4	23	32	8	45	56	78
Step 4	23	8	32	45	56	78
Step 4	8	23	32	45	56	78
Step5	8	23	32	45	56	78
Step 4	8	23	32	45	56	78
Step 4	8	23	32	45	56	78
Step 4	8	23	32	45	56	78
Step 5	8	23	32	45	56	78
Step 4	8	23	32	45	56	78
Step 4	8	23	32	45	56	78
Step5	8	23	32	45	56	78
Step 4	8	23	32	45	56	78
Step 5	8	23	32	45	56	78
Step 4	8	23	32	45	56	78

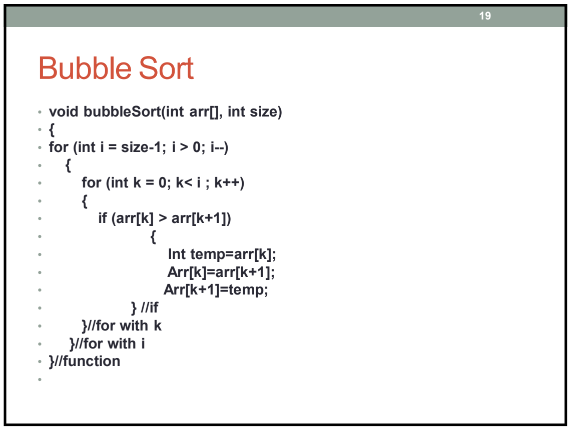
🟦 Comparison

🟩 Sorted Sublist



Insertion Sort

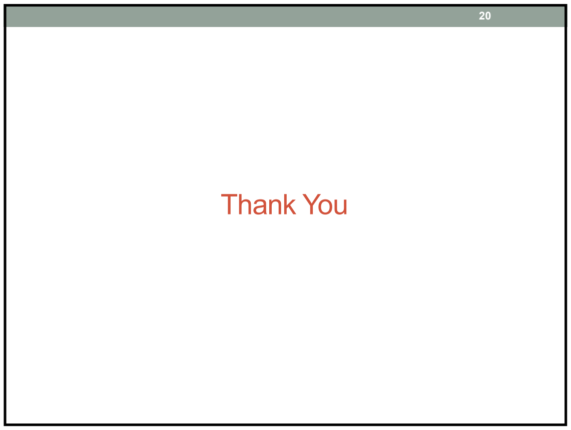
```
void insertionSort(int arr[], int n)
{   int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] < key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }}
```



19

Bubble Sort

```
void bubbleSort(int arr[], int size)
{
for (int i = size-1; i > 0; i--)
   {
      for (int k = 0; k< i ; k++)
      {
         if (arr[k] > arr[k+1])
                  {
                     Int temp=arr[k];
                     Arr[k]=arr[k+1];
                    Arr[k+1]=temp;
               } //if
      }//for with k
    }//for with i
}//function
```



20

Thank You