Date
Page

Assignment 2

①
```
void search (int arr[] int n , int key)
{
    for( int i = 0 ; i<n ; i++)
    {
        if( arr[i] == key)
        { cout<< " found " <<endl;
            break;
        }
        else if (arr[i] > key)
            {   break;
            }
    }
}
```

② Iterative pseudo code of Insertion sort ;

```
void iterative( int arr[] , int n)
{ int i, j , key;
    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;
    }
}
```

```
void recursive (int arr [], int n)
{
    if (n <= 1)
      { return ;
      }
    recursive (arr, n-1).
      key = arr[n-1]
      j = n-2;
    while (j>0 && arr [j] > key)
      {
        arr[j+1] = arr[j];
          j--;
      }
    arr[j+1] = last;
}
```

(3)

Algorithm	time complexity			Space complexity
	Best T.C.	Worst T.C	Avg	
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Count	$O(n+K)$	$O(n+K)$		$O(K)$
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Quick	$O(n\log n)$	$O(n^2)$	$O(n\log n)$	$O(\log n)$
Radix	$O(nK)$	$O(nK)$	$O(nK)$	$O(n+K)$
Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$
Linear	$O(1)$	$O(n)$	$O(n)$	$O(N)$
Binary	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$

```
void recursive (int arr [], int n)
{
    if (n <= 1)
      { return ;
      }
    recursive (arr, n-1).
      key = arr[n-1]
      j = n-2;
    while (j>0 && arr [j] > key)
    {
        arr[j+1] = arr[j];
          j--;
    }
    arr[j+1] = last;
}
```

(3)

Algorithm	time complexity			Space complexity
	Best T.C.	Worst T.C	Avg	
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Count	$O(n+k)$	$O(n+k)$		$O(k)$
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Quick	$O(n\log n)$	$O(n^2)$	$O(n\log n)$	$O(\log n)$
Radix	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$
Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$
Linear	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Binary	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$

(4) Algorith

Selectio
Insert
Coun
merg
Qui
Ra
H
L
R

(5)

(4)

Algorithm	Inplace	stable	online sorting
Selection	Yes	NO	NO
Insertion	Yes	Yes	Yes
Count	NO	yes	NO
merge	NO	Yes	NO
Quick	NO	yes	Yes
Radix	NO	Yes	NO
Heap	NO	Yes	NO
Linear			
Binary		yes	
Bubble.	Yes	Yes	NO

(5) Itrative:-

```
int Binary (int arr[] ,int k)
{
    int l =0;
    int h =arr.size[ ]-1;
    while( l<=h )
    {
        int mid = l + (h-l)/2 ;
        if (arr[mid ]= k )
        {  return mid
        }
    else if(arr[mid] <k)
        {   l= mid+1 ;
        }
```

(4)

Algorithm	Inplace	stable	online sorting
Selection	Yes	NO	NO
Insertion	Yes	Yes	Yes
Count	NO	yes	NO
merge	NO	Yes	NO
Quick	NO	yes	Yes
Radix	NO	Yes	NO
Heap	NO	Yes	NO
Linear			
Binary		yes	
Bubble.	Yes	Yes	NO

(5) Iterative:-

```
int Binary(int arr[], int k)
{
    int l = 0;
    int h = arr.size[] - 1;
    while(l <= h)
    {
        int mid = l + (h-l)/2;
        if (arr[mid] = k)
        {   return mid;
        }
    else if(arr[mid] < k)
        {   l = mid+1;
        }
```

```
else
{ h = mid - 1
  }
  }
  return -1
  }
```

T.C = O(logn)
S.C = O(1)

→ Recursive :-

```
bool Binary(int arr[], int l, int r, int K)
{
  if (l > r).
    return false;
  int mid = l + (r-l)/2;
  if (arr[mid] = = K)
      return true;

  else if (arr[mid] > key)
      Binary(arr, l, mid-1, key);

  Binary (arr, mid+1, r, K);
}
```

T.C. = O(logn)
S.C = O(1)

(6) The recurrence relation for binary recursive search can be expressed as:

$$T(n) = T(n/2) + O(1)$$

(5) The Best case scenario for quick sort is when the chosen pivot element divides the array into two approximately equal halves. (T.C. = $O(n \log n)$)
As for the worst case scenario is when chosen ~~pio~~ pivot element divides the array into 2 unbalanced halves.
(T.C = $O(n^2)$).

(7) In an array, an inversion occurs when two elements are not ~~of~~ their sorted ~~array~~ order.
for eg: Consider an array A, if ~~there~~ there are two indices i ~~and~~ and j such that i<j and ~~A~~ A[i] > A[j], then the pair (A[i], A[j]) forms ~~and~~ an inversion

Counting inversion merge sort.

(6) The recurrence relation for binary recursive search can be expressed as:

$$T(n) = T(n/2) + O(1)$$

(5) The Best case scenario for quick sort is when the chosen pivot element divides the array into two approximately equal halves. (T.C. = $O(n \log n)$)
As for the worst case scenario is when chosen ~~pio~~ pivot element divides the array into 2 unbalanced halves.
(T.C = $O(n^2)$).

(7) In an array, an inversion occurs when two elements are not in sorted ~~arry~~ order.
for eg: Consider an array A, if ~~there~~ there are two indices i ~~arry~~ and j such that i<j and ~~A~~ A[i] > A[j], then the pair (A[i], A[j]) forms ~~and~~ an inversion

Counting inversion merge sort.

```
int mergesort(int arr[], int temp[], int l, int r)
{ int mid, inv = 0;
  if (r>l)
  { mid = (r+l)/2;
  inv = inv + mergesort (arr, temp, l, mid);
  inv = inv + mergesort (arr, temp, mid+1, r);
  inv = inv + merge (arr, temp, l, mid+1, r);
  }
  return inv;
}
int merge(int arr[], int temp[], int l, int mid, int r)
{
  int i, j, k;
  int inv = 0;
  i = l;
  j = mid;
  k = l;
  while((i<mid-1) && (j<=r))
  { if (arr[i] <= arr[j])
    { temp[k++] = arr[i++];
    }
  else
    {
      temp[k++] = arr[j++];
      inv = inv + [mid-i];
    }
  }
```

```
→ int mergesort(int arr[], int temp[], int l, int r)
  { int mid, inv = 0;
      if (r>l)
      { mid = (r+l)/2;
    inv = inv + mergesort (arr, temp, l, mid);
    inv = inv + mergesort (arr, temp, mid+1, r);
    inv = inv + merge (arr, temp, l, mid+1, r);
      }
    return inv;

  }

int merge(int arr[], int temp[], int l, int mid, int r)
{
    int i, j, k;
    int inv = 0;
    i = l;
    j = mid;
    k = l;
  while((i<mid-1) && (j<=r))
  { if (arr[i] <= arr[j])
      { temp[k++] = arr[i++];
      }
    else
      {
        temp[k++] = arr[j++];
          inv = inv + [mid-i];
        }

  }
```

```
while(i <= mid+1)
{ temp[k++] = arr[j++] ;
}
For( i=l ; i<=r ; i++)
{ arr[ i]= temp[i]}
}
return inv ;
}
int merges( int arr[ ] , int size)
{
int temp[size] ;
return mergesort( arr, temp , 0, size-1);
}
```

13 Modified version of bubble sort that can resolve the problem is :-

```
void bubble (int arr[])
{
int n= arr.size[] , i=0, j=0;
bool swape;
For (i=0; i<n-1 ; i++)
{
swape = false;
For( j=0 ; j<n-i-1 ; j ++)
{ if( arr[j] > arr[ j+1]
{
swap(arr[j], arr[j+1]);
swap = true;
}
}
```

```
while(i <= mid+1)
{ temp[k++] = arr[j++];
  }
  For(i=l; i<=r; i++)
   {
     arr[i]= temp[i];
   }
   return inv;
   }
int merges(int arr[·], int size)
  {
    int temp[size];
    return mergesort(arr, temp, 0, size-1);
  }
```

13 Modified version of bubble sort that can resolve the problem is :-

```
void bubble(int ar[])
  {
    int n = arr.size[], i=0, j=0;
    bool swape;
    For(i=0; i<n-1; i++)
      {
        swape = false;
        for(j=0; j<n-i-1; j++)
        { if(arr[j] > arr[j+1])
          {
            swap(arr[j], arr[j+1]);
            swap = true;
          }
        }
```

```
if (! swape)
{
    break;
}
}
```

12 stable version of selection sort :-

```
void selection (int arr[ ])
{
    int n = arr.size(), i=0;
    for (i=0; i<n-1; i++)
    {
        int min = i;
        for (int j= i+1 ; j<n; ++j)
        {
            if (arr[j] < arr[min])
            {
                min = j;
            }
        }
        int m= arr[min];
        while (min> i)
        { arr[min] = arr[min-1];
            min--;
        }
        arr[i] = m;
    }
```

```
if (! swape)
  {
    break;
  }
}
```

12 stable version of selection sort :-

```
void selection (int arr[ ])
  {
    int n = arr.size(), i=0;
    For (i=0; i<n-1; i++)
      {
        int min = i;
        For (int j = i+1; j<n; ++j)
          {
            if (arr[j] < arr[min])
              {
                min = j;
              }
          }
        int m = arr[min];
        while (min > i)
        { arr[min] = arr[min-1];
          min--;
        }
        arr[i] = m;
      }
```