CS1100
Introduction to Programming

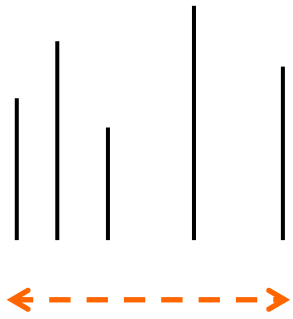
Sorting

Random Q

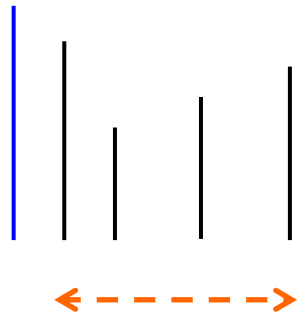
0	1	2	3	4	5	6	7
93	83	73	63	53	43	33	23

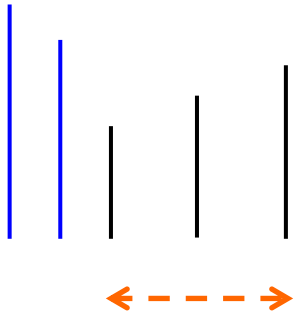
Let X = 43; assume that Binary Search algorithm is used.

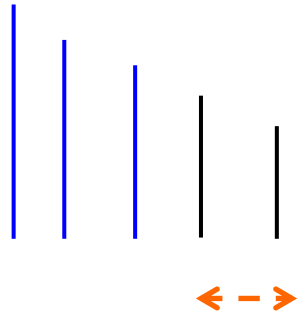
What are the values compared to before finding 43?

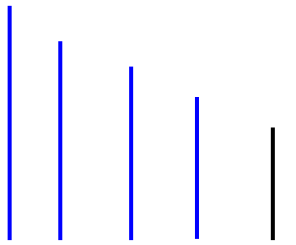


Sort, in descending order









Sorting an Array of Numbers (storing Marks)

- Problem: Arrange the marks in **decreasing** order starting with the maximum
  - Array has 100 elements (say)
- One approach
  - Find the **maximum** value in ***marks*[0]** … *marks*[99]
  - Remember the index *i* where it occurred
  - Exchange (values of) *marks*[0] and *marks*[*i*]
  - Find the **maximum** value in ***marks*[1]** to *marks*[99]
  - exchange *marks*[1] and *marks*[*i*]
  - . . . do this till marks[98]

Swap Array Elements with indices i and j

```
void swap (int array[ ], int i, int j)
{
int temp;
temp = array[i];
array[i] = array[j];
array[j] = temp;
}
```


Where is the Highest Number?

- Given an array of *n* elements, a starting index *(start)*, find out where the largest element lies beyond and including *start*.

```
int FindMaxIndex (int array[ ], int start, int arraySize){
  int i, index, max;
  index = start; max = array[start];
  for (i = start; i < arraySize;  i++)
  {
     if (array[i] > max){
          max = array[i];
index = i;
     } /* End if */
  } /* End for */
 return index;
}
```


Selection Sort

swap is an function that passes array by reference.

The *last* element need not be tested

```
for (i=0, i <= n - 2, i++)
{
   int maxIndex = FindMaxIndex(marks, i, n);
   if (maxIndex != i) swap(marks, i, maxIndex);
}
```



```
void swap (int array[ ], int i, int j)
{
int temp;
temp = array[i];
array[i] = array[j];
array[j] = temp;
}

Swap(A, i, j);
```

```
void swap (int x, int y)
{
int temp;
temp = x;
x = y;
y = temp;
}

Swap(A[i], A[j]);
```

RANDOM Q

Selection Sort as a Function

```
void selectSort(int array[ ], int n)
{
  int maxIndex, i;
  for (i = 0; i <= n – 2; i++)
  {
maxIndex = FindMaxIndex(array, i, n);
if (maxIndex != i) swap(array, i, maxIndex);
  }
}

int main(){
 const int LEN=6;
 int array1[] = {23, 31, 42, 11, 16, 7};
 selectSort(array1, LEN);
 for (int i = 0; i < LEN; i++) printf("%d ", array1[i]);
}
```


Extra Q

3 9 11 2 6 10

Write the array contents for the first and second iteration of Selection Sort, in descending order

Random Q

```
int FindMaxIndex (int array[ ], int start, int arraySize){
  int i, index, max;
  index = start; max = array[start];
  for (i = start; i < arraySize;  i++)
  {
    if (array[i] > max){
       max = array[i];
      index = i;
    }}
 return index;}
```

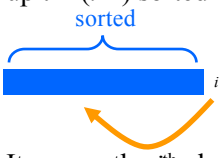
```
int FindMaxIndex (int *array, int start, int arraySize){
  int index, max,*last = &array[arraySize-1];
  index = start; max = *array;
  for (; array != last; array++)
  {
    if (*array > max){
       max = *array;
       index = i;
    }} return index;
}
```


An Example

0	1	2	3	4	5	6	7	*i*	maxIndex
2	1	7	5	8	3	6	4	0	4
8	1	7	5	2	3	6	4	1	2
8	7	1	5	2	3	6	4	2	6
8	7	6	5	2	3	1	4	3	3
8	7	6	5	2	3	1	4	4	7
8	7	6	5	4	3	1	2	5	5

Insertion Sort

- Insertion sort also scans the array from left to right
- When it looks at the i^{th} element, it has elements up till (i-1) sorted



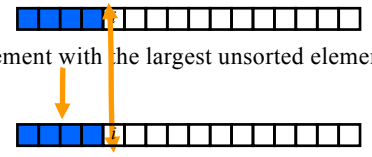
- It moves the i^{th} element to its correct place by shifting the smaller elements to the right

An Example

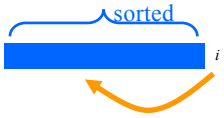
0	1	2	3	4	5	6	7	*i*	# of comp
2	1	7	5	8	3	6	4	1	1
2	1	7	5	8	3	6	4	2	2
7	2	1	5	8	3	6	4	3	3
7	5	2	1	8	3	6	4	4	4
8	7	5	2	1	3	6	4	5	3
8	7	5	3	2	1	6	4	6	5

Selection Vs Insertion Sort

- Scanning from left to right
- Selection sort
  - Swaps the i^{th} element with the largest unsorted element
- Insertion sort
  - Inserts the ith element into its proper place

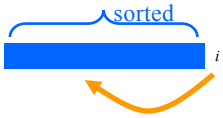


InsertMax Function: Descending order




```
void  InsertMax (int array[ ], int index){
int i = index;
int valueAtIndex = array[index];
while(i > 0 && array[i-1] < valueAtIndex) {
        array[i] = array[i-1];     /*shift right*/
        i--;
}
      array[i] = valueAtIndex;
}
```

Complexity of InsertMax



- If the i^{th} element is in sorted order (smaller than the sorted set), no shift is done
- The maximum number of shifts is (i-1)
- Complexity
  - worst case O(i)
  - best case O(1) – constant time

Insertion Sort Function

```
void InsertionSort(int array[ ], int size){
    int i;
    for(i = 1; i <= size – 1; i++)
  InsertMax(array, i);
}
```

- Complexity
  - best case O(n)
  - worst case O($n^2/2$) = O(n^2)

Selection Vs Insertion

- Selection sort always does the same number of computations irrespective of the input array
- Insertion sort does less work if the elements are partially sorted
  - when the i^{th} element is in place, it does not have to shift any elements – constant time
- If the input is already sorted, Insertion sort merely scans the array left to right – confirming that it is sorted
- On the average, Insertion sort performs better than Selection sort

MISC AND OPTIONAL

Exercise

- Given an array of strings, called *names*, and an array of marks, called *marks*, such that *marks*[*i*] contains the marks of *names*[*i*]
  - sort the two lists in decreasing order of marks
  - sort the two lists in alphabetic order of names
    - figure out how to compare two names to decide which comes first.

Lexicographic (Dictionary) Ordering

- Badri < Devendra
- Janak < Janaki
- Shiva < Shivendra
- Seeta < Sita

- Based on the ordering of characters
  - A < B … < Y < Z < a < b < c < . . . < y < z

upper case before lower case

Lexicographic Ordering

- What about blanks?
  - "Bill Clinton" < "Bill Gates"
  - "Ram Subramanian" < "Ram Subramanium"
  - "Ram Subramanian" < "Rama Awasthi"
- In ASCII the blank (code = 32) comes before all other characters. The above cases are taken care of automatically.
- Exercise: Look up ASCII codes on the web.

Lexicographic Ordering

- What if two names are identical?
- There is a danger that the character arrays may contain some unknown values beyond '\0'
- Solutions
  - One could begin by initializing the arrays to blanks before we begin.
  - One could explicitly look for the null character '\0'
  - When the two names are equal it may not matter if either one is reported before the other. Though in stable sorting there is a requirement that equal elements should remain in the original order.

Comparing Strings (char Arrays)

- Given two strings *A*[*i*][] and *A*[*j*][] of length *n*, return the index of the string that comes earlier in the lexicographic order

```
int strCompare(char A[ ][MAX_SIZE], int i, int j, int
MAX_SIZE){
int k=0;
while ((A[i][k] == A[j][k]) && k<MAX_SIZE) k++;
if (A[i][k] == '\0') return i;
if (A[j][k] == '\0') return j;
if (A[i][k] < A[j][k]) return i;
else return j;
}
```

Skip common characters if any

If one string is *prefix* of the other return that

Built-in String Comparison

- ***#include*** <string.h>
- ***int*** *strcmp*(***const char*** *s1, ***const char*** *s2);
- ***int*** *strncmp*(***const char*** *s1, ***const char*** *s2, size_t n);
- ***int*** *strcmp*(char*, char*) – compares two strings (char arrays)
- The return values are:
  - 0 – If both strings are equal
  - >0 – If first string is lexicographically greater than second
  - <0 – If second string is lexicographically greater than first

Pointers - address of char array – we will look at them later

Sorting String Arrays

- Modify *InsertionSort* to sort array *names*[ ] of names
- In the exercise where *names*[ ] and *marks*[ ] have to be sorted in concert, modify the sorting algorithm to
  - compare in one array
    - *names*[ ] for alphabetic order
    - *marks*[ ] for decreasing marks or
  - move elements of both

Compact structures to hold both to be explored later

Printing a Reversed String

```
main( ) {
int i = 4;
char c;
do{
c = "hello"[i];
printf("%c",c);
i --;
    }while(i >= 0);
printf("\n");
}
```