CMSC11: SEARCHING AND SORTING ALGORITHMS
by: Maverick Crisostomo

Objectives:
At the end of the lesson, you should:

1. Be able to explain linear search and binary search;
2. Be able to implement some simple sorting algorithms; and
3. Have learned the concepts behind running time of algorithms.

LINEAR SEARCH vs BINARY SEARCH

LINEAR SEARCH	**BINARY SEARCH**
a.k.a **sequential search**	Requires the list to be sorted.

Pseudocode:
 	
For each item in the list:
Check to see if the item you're looking for matches the item in the list.
If it matches.
Return the location where you found it (the index).
If it does not match.
Continue searching until you reach the end of the list.
 	
If we get here, we know the item does not exist in the list. Return -1.
	Pseudocode:
	
	while ((more than one item in list) and (haven't yet found target))
	look at the middle item
	if (middle item is target)
	have found target
	else if (target < middle item)
	list = first half of list
	else (target > middle item)
	list = last half of list
	end while
	if (have found target)
	location = position of target in original list
	return location as the result

int sequentialsearch(int a[10],int x)
{
int ptr=0;
n=10;
while (ptr<n)
if (a[ptr] != x)
ptr++;
else
return(1);
return(0);
}
	int binarysearch(int a[10], int x)
	{
	int lower, upper, middle;
	lower=0;
	n=10;
	upper=n-1;
	while (lower<=upper) {
	middle=(lower+upper)/2;
	if (x>a[middle]) lower=middle+1;
	else if (x<a[middle]) upper=middle-1;
	else return(1);
	}
	return(0);
	}

Worst Case complexity: O(n) Best Case complexity: O(1) Average Case complexity: O(n)	Worst Case complexity: O(log n) Best Case complexity: O(1) Average Case complexity: O(log n)

SORTING ALGORITHMS

are algorithms for arranging elements of a list in a certain order. Here are some simple sorting algorithms: the bubble sort, insertion sort and selection sort.

Bubble Sort

The bubble sort works by comparing each item in the list with the item next to it, and swapping them if required. The algorithm repeats this process until it makes a pass all the way through the list without swapping any items (in other words, all items are in the correct order). This causes larger values to "bubble" to the end of the list while smaller values "sink" towards the beginning of the list.

```
void bubbleSort(int numbers[], int array_size)
{
  int i, j, temp;

  for (i = (array_size - 1); i >= 0; i--)
  {
    for (j = 1; j <= i; j++)
    {
      if (numbers[j-1] > numbers[j])
      {
        temp = numbers[j-1];
        numbers[j-1] = numbers[j];
        numbers[j] = temp;
      }
    }
  }
}
```

Insertion Sort

It inserts each item into its proper place in the final list.

```
void insertionSort(int numbers[], int array_size)
{
  int i, j, index;

  for (i=1; i < array_size; i++)
  {
    index = numbers[i];
    j = i;
    while ((j > 0) && (numbers[j-1] > index))
    {
      numbers[j] = numbers[j-1];
      j = j - 1;
    }
    numbers[j] = index;
  }
}
```

Selection Sort

The selection sort works by selecting the smallest unsorted item remaining in the list, and then swapping it with the item in the next position to be filled.

```
void selectionSort(int numbers[], int array_size)
{
  int i, j;
  int min, temp;

  for (i = 0; i < array_size-1; i++)
  {
    min = i;
    for (j = i+1; j < array_size; j++)
    {
      if (numbers[j] < numbers[min])
        min = j;
    }
```

```
    temp = numbers[i];
    numbers[i] = numbers[min];
    numbers[min] = temp;
  }
}
```

RUN TIME ANALYSIS of ALGORITHMS

a theoretical classification that estimates and anticipates the increase in running time (or run-time) of an algorithm as its input size (usually denoted as "n") increases.

Complexity Classes

Description	**O-notation**
Constant	O(1)
Logarithmic	O(log n)
Linear	O(n)
N log n	O(n log n)
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Polynomial	$O(n^k)$, k >= 1
Exponential	$O(a^n)$, a > 1

Practice Exercise:

Trace the elements in the array after every iteration using the three sorting algorithms discussed. Use the ff. inputs for the array:

1. [5, 7, 4, 3, 2, 9, 10]
2. [10, 9, 3, 5, 6, 8, 1, 4, 2, 7]