Sorts & Searches

Introduction

In this unit we are going to look at sorting and searching algorithms.

Searches

Searches are used to find a value in a list. When a value is found, a search will return the index of the value, and when a value is not found it will return -1.

Linear Search

A **linear search** is a sort that can work on both ordered and unordered lists. The search starts at the beginning of the list and moves through it one item at a time until it either finds the value or runs outof data.

Notes:

- Linear searching is slow
- Linear searches work on ordered and unordered data.

Pseudo-code:

```
found =-1
for x = 0 to end
        if data[x] =value
                found = x
                break
return found
```

Example 1:

Value	4	7	6	9	3
Location	0	1	2	3	4

searching for: 6

Result: 2
Location Check Order: 0, 1, 2
Locations Checked: 3

Example 2:

Value	4	7	6	9	3
Location	0	1	2	3	4

searching for: 8

Result: -1
Location Check Order: 0, 1, 2, 3, 4
Locations Checked: 5

Binary Search

A **binary search** is a sort that can only work on ordered lists. The search works by eliminating ½ of the items left to be searched on every check. For example, for a list of 1000 values, 500 would be eliminated in the first check and 250 of the remaining 500 would be eliminated on the second check.

Notes:

- Binary searching is fast
- Binary searches can only be made on ordered data.

Pseudo-code:

```
start = 0
end = last index
while start <=end
        int check = (start+end)/2
        if data[check] == value
                return check
        else data[check] > value
                end = check -1
        else
                start = check +1
return -1
```

Example 1:

Value	1	4	5	8	9	12	23	45	66
Location	0	1	2	3	4	5	6	7	9

searching for: 45

Result: 7
Location Check Order: 4, 7
Locations Checked: 2

Example 2:

Value	1	4	5	8	9	12	23	45	66
Location	0	1	2	3	4	5	6	7	9

searching for: 6

Result: -1
Location Check Order: 4, 1, 2, 3
Locations Checked: 4

Sorts

Sorts are used to put data in order.

Selection Sort

Selection sort starts at index 0 and works its way to the end of the list. At each index the sort will find the location of the smallest value from x to the end of the list and then swap the values at x and where the smallest value is. The effect of this swap is putting the correct value into location x.

Notes:

- Selection Sorts are slow.

Pseudo-code:

```
for a=0 to end
        minIndex = a
        for b = a+1  to end
                if data[b] < data[minIndex]
                        minIndex = b
        swap values at minIndex and a
```

Example:

Value	4	7	6	9	3
Location	0	1	2	3	4

Data At Start	4	7	6	9	3
After 1st Pass	3	7	6	9	4
After 2nd Pass	3	4	6	9	7
After 3rd Pass	3	4	6	9	7
After 4th Pass	3	4	6	7	9

Insertion Sort

Insertion sort starts at index 1 and works its way to the end of the list. At each index it will move the value to the left if needed so that the values in locations 0 to x are in order. After processing the last index all of the values will be in order.

Notes:

- Insertion sorts are faster than selection sorts, but are still fairly slow.
- Insertion sort is the only sort that works really fast on data sets that are already sorted or almost sorted.

Pseudo-code:

```
for i = 1 to end
        temp = data[i]
        j = i
        while j > 0 && data[j-1] > temp
                data[j] = data[j-1]
                j = j -1
        data[j] = temp
```

Example:

Value	4	7	6	9	3
Location	0	1	2	3	4

Data At Start	4	7	6	9	3
After 1st Pass	4	7	6	9	3
After 2nd Pass	4	6	7	9	3
After 3rd Pass	4	6	7	9	3
After 4th Pass	3	4	6	7	9

Merge Sort

Merge Sort works by cutting the list in ½ and then cutting them those lists in ½ and so on until there are N lists of size 1. The lists are then merged back together and put in the correct order starting with the ones that were most recently split. At the last stage of the algorithm you have to put 2 ordered lists together, each containing ½ of the originals list's data.

Notes:

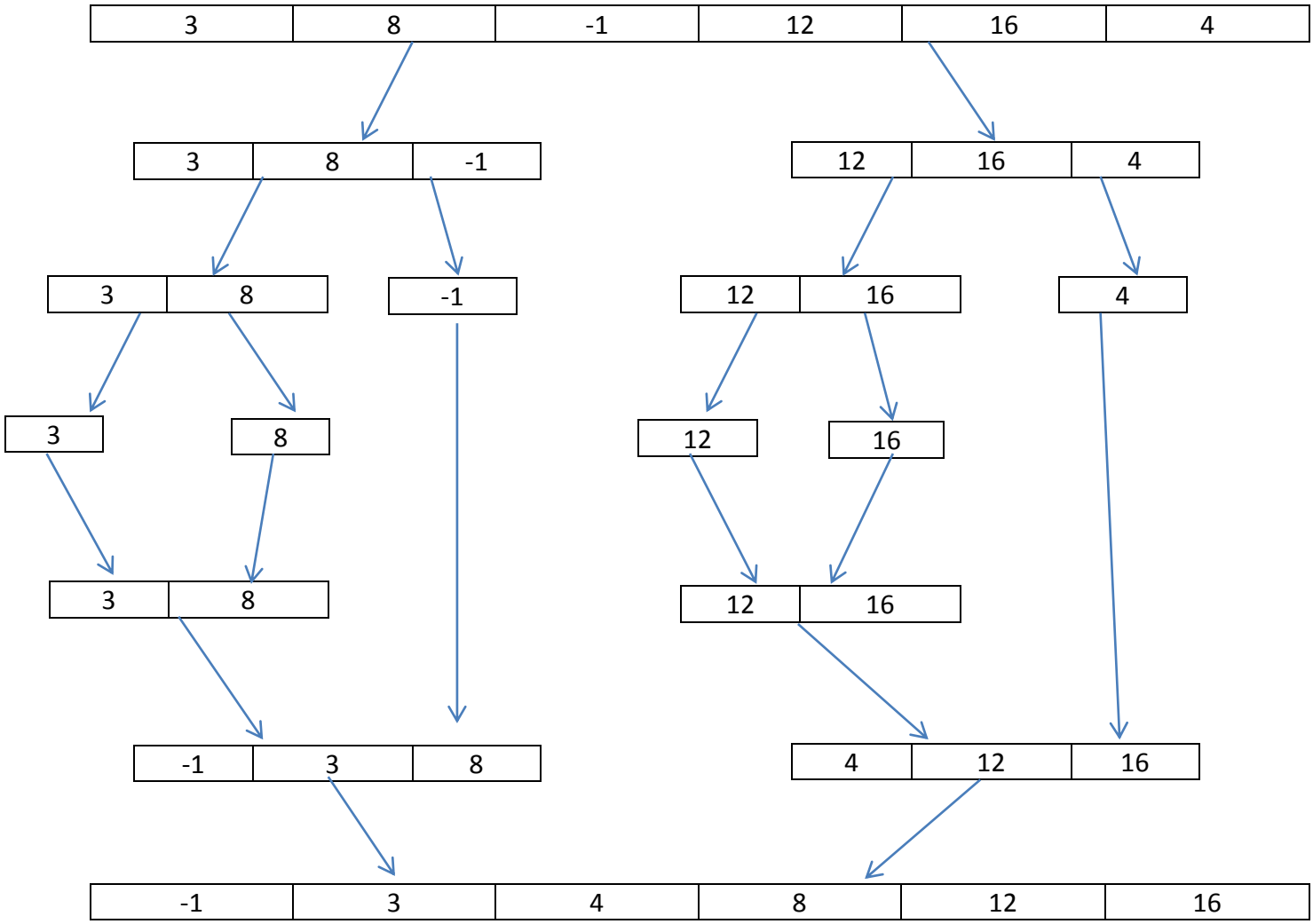
- Merge Sort is fast.
- The only time merge sort is not as fast as insertion sort is when the data is already almost in order.
- Requires a temp array that is of the same type and size of the array being sorted. The temp array should be a static attribute.

- When the split is not even the extra will always go left or right. You decide which side to always give it to.

Pseudo-code:

```
if to == from
        return
middle = (from+to )/2
mergeSort data, from, middle
mergeSort data, middle+1, to

i = from
j = middle+1
k =from

while i<=middle && j <= to
        if data[i] < data[j]
                tempData[k++] = data[i++]
        else
                tempData[k++] = data[j++]

while i <= middle
        tempData[k++] = data[i++]

while j<=to
        tempData[k++] = data[j++]

for k = from to to
        data[k] = tempData[k]
```

Example:



Grading Algorithms

There are many ways to accomplish the same task and some will be faster than others. For example someone doing 8 loads of laundry at home, using one washer and dryer, may spend all day cleaning clothes. Another individual does his/her 8 loads at a laundry mat might only need an hour or 2 if there are 8 washers and dryers available.

The first algorithm uses fewer resources, but is slower. The second algorithm is faster, but requires more resources. With computers usually there is a tradeoff; slower algorithms usually require more time but less resources, where faster algorithms take less time but use more resources.

We are only going to be grading algorithms based on speed, but it is important to keep in mind how resources are used.

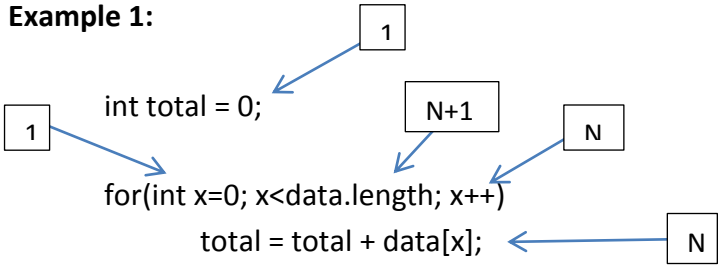
Statement Counting

Statement counting is very basic form of grading algorithms where you try to determine the exact number of statements executed to perform a task. In statement counting when there is a list its size is denoted with a variable, like N.

How to Count:

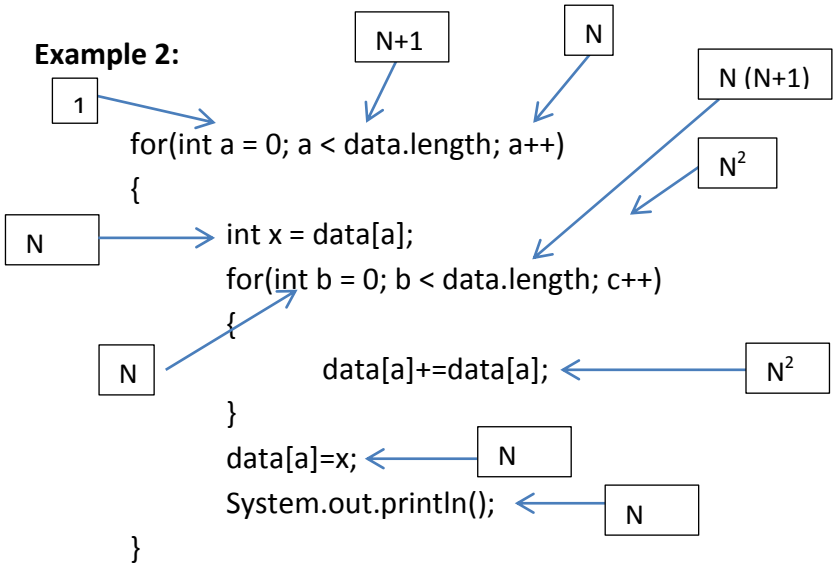
- When a statement runs 1 time you count it as 1.
- When a statement runs 1 time for each value in a list you count it as N.
- When a statement runs 1 time for each item in a list and then one more time you count it as N+1.
- If you have a loop inside of another loop and each loop runs N times, most the statements in the inner for loop run N^2 times

Example 1:



1 + 1 + (N + 1) + N + N = 3N + 3

Example 2:



$1 + (N+1) + N + N + N + (N^2+N) + N^2 + N^2 + N + N$

Summation from 1 to N

The value of a summation from 1 to N is always:

$(n(n+1))/2$

Work Sheets:

Searches & Sorts Worksheet
Statement Count Worksheet