Selection Sort

Lexicographic
= alphabetical dictionary order

Loops — Start : 34, 12, 8, 15, 1

1st — Outer : 1, 12, 8, 15, 34

2nd — Outer : 1, 8, 12, 15, 34

3rd — Code to check if loop is sorted :

```
if (key == A[index]) {
    found = true;
    location = index;
}
```

Bubble Sort

Loops: Start : 34, 12, 8, 15, 1

1st — 12, 8, 15, 1, 34

8, 12, 1, 15, 34

1, 8, 12, 15, 34

Current largest to the bottom.
Looks at neighbouring items and sort out of place.

```
import java.util.Arrays;

int ARRAY[] = {7, 1, 9, 4};
for(int number = ARRAY) {
  SysOut("Number = " + number) }
Arrays.sort(ARRAY); // sorting array
```

Bubble Sort

```
public static void bubbleSort (int[] list)
{
      int temp;    int pass = 0;

      boolean anotherPassNeeded = true;
      int currentBottom = list.length;

      while (anotherPassNeeded)
      {
            anotherPassNeeded = false;

            for (int curr = 0; curr < (currentBottom - 1); curr++)
                  if (list[curr + 1] < list[curr])
                  {
                        temp = list[curr];
                        list[curr] = list[curr + 1];
                        list[curr + 1] = temp;

                        anotherPassNeeded = true;
                  }

            currentBottom = currentBottom - 1;

      }
  }
}
```

---

class method. Not called by object. Called by main

Selection Sort

```
private static void selectionSort (int[] list)
{
      int minIndex, index, j;
      int temp;  int pass = 0;

      for (index = 0; index < list.length-1; index++)
      {
            minIndex = index;
            for (j = minIndex+1; j < list.length;  j++)
                  if (list[j] < list[minIndex])
                        minIndex = j;

            if (minIndex != index)
            {
                  temp = list[index];
                  list[index] = list[minIndex];
                  list[minIndex] = temp;
            }

      }
  }
}
```

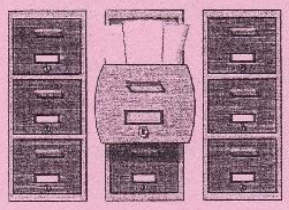
For names, use compareTo < 0

for descending order

3 lines of code to do swap

Selection sort

Computer Science 205 - Searching and Sorting Reference



Linear Search

linearSearch is very slow

```
private static int linearSearch (int[] A, int key)
{
      boolean found = false; int index = 0;  int location = -1;

      while ((! found) && (index < A.length))
      {
            if (key == A[index])
            {
                  found = true;
                  location = index;
            }
            index++;
      }
      return location;
}
```

starts at index 0

Binary Search

```
private static int binarySearch (int[] list, int key)
{
      int first = 0, last = list.length-1, middle, location;

      boolean found = false;

      do
      {
            middle = (first + last) / 2;

            if (key == list[middle])
                  found = true;
            else if (key < list[middle])
                  last = middle - 1;
            else
                  first = middle + 1;

      } while ( (! found) && (first <= last) );

      location = middle;

      return (found ? location : -1);
}
```

Binary Search

Is much more efficient than Linear Search.

Needs to be sorted first in ascending order

1) Take the first and last array indicies to calculate the middle index

$$\text{middle index} = \frac{(\text{first} + \text{last})}{2}$$

Integer division = $\frac{0+1}{2} = 0$

(Integers always round down
0.5 = 0
1.5 = 1
2.333 = 2)

2) Resets first or last depending on where middle item currently is with respect to the key (item we're looking for)

Number of Searches

Binary: 1 search at best (average + worst case)
$\log_2 n$ where n is array size

Linear: $n/2$ is average
n searches is worst
1 search is best

e.g. Array size = 1000

$\log_2 n$

$2^9 = 512$

$2^{10} = 1024$

Therefore binary search takes 10 searches average

The Binary Search

A Much Better Searching Algorithm

```
private static int binarySearch (int[] list, int key)
{
        int first = 0, last = list.length-1, middle, location;

        boolean found = false;

        do
        {
                middle = (first + last) / 2;

                if (key == list[middle])
                        found = true;
                else if (key < list[middle])
                        last = middle - 1;
                else
                        first = middle + 1;

        }  while ( (! found) && (first <= last) );

        location = middle;

        return (found ? location : -1);
}
```

resetting last or first
thus first = middle + 1
in a do-while loop

if (found)
return location;
else
return -1;

when the thing we are looking for is not there

it is not there

0	1	2	3	4
34	12	8	15	1

Linear search for key 15 = 4 checks to find
-1 = 5 checks because it goes through all to find it is not there

Given array size 'n', on average n/2 checks to find the key

first	last	middle
0	4	2
3	4	3
4	4	4

3 checks

Best case = 1 check
Worst case = n checks

34

(first + last) / 2

Computer Science 205
Sorting and Searching Review

Given the following declarations :

```
int[] list = {8, 12, 3, -6, 9, 1, 0, 5}
int key;
```

Answer the questions which follow:

1. After sorting the array in ascending order (least to greatest), how many checks will you have to make in order to locate the following `key` values using the linear search algorithm:

   a. 8
   b. –6
   c. 15

2. After sorting the array in ascending order (least to greatest), how many checks will you have to make in order to locate the following `key` values using the binary search algorithm:

   a. 8
   b. –6
   c. 15

3. Using the bubble sort algorithm with ascending order, show what array variable `list` looks like after each pass through the outer loop completes.

4. Using the bubble sort algorithm with descending order, show what array variable `list` looks like after each pass through the outer loop completes.

5. Using the selection sort algorithm with ascending order, show what array variable `list` looks like after each pass through the outer loop completes.