

# **SORTING AND SEARCHING**

**Lab 22**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Searching

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **Linear / Sequential Search**

**The Linear Search searches through a list one element at time looking for a match. The index position of a match is returned if found or -1 is returned if no match is found.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Linear with Primitives

```
int linearSearch(int[] stuff, int val)
{
    for(int i=0; i< stuff.length; i++)
    {
        if (stuff[i] == val )
            return i;
    }
    return -1; //returns -1 if not found
}
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Linear with Objects

```
int linearSearch(Comparable[] stuff,  
                  Comparable item)  
{  
    for(int i=0; i<stuff.length; i++)  
    {  
        if (stuff[i].compareTo(item)==0)  
            return i;  
    }  
    return -1; //returns -1 if not found  
}
```

© A+ Computer Science - www.apluscompsci.com

**open  
linearsearch.java  
linearsearchtester.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

**1/2**

# **Binary Search**

**0100100101**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# BinarySearch

**The Binary Search works best with sorted lists. The Binary search cuts the list in half each time it checks for the specified value. If the value is not found, the search continue in the half most likely to contain the value.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

```
int binarySearch (int [] stuff, int val )  
{  
    int bot= 0, top = stuff.length-1;  
    while(bot<=top)  
    {  
        int middle = (bot + top) / 2;  
        if (stuff[middle] == val) return middle;  
        else  
            if (stuff[middle] > val)  
                top = middle-1;  
            else  
                bot = middle+1;      BinarySearch  
    }  
    return -1;  
}
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

```
int[] stuff = {1,6,8,10,14,22,30,50};
```

**0 + 7 = 7 / 2 = 3  
stuff[3] = 10**

**4 + 7 = 11 div 2 = 5  
stuff[5] = 22**

**6 + 7 = 13 div 2 = 6  
stuff[6] = 30**

## BinarySearch

**If you are searching for  
25, how many times will  
you check the stuff?**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **Binary Search ShortCut**

**Given a list of N items.**

**What is the next largest power of 2?**

**If N is 100, the next largest  
power of 2 is 7.**

**Log<sub>2</sub>(100) = 6.64386**

**2<sup>7</sup> = 128.**

**It would take 7 checks max to find if an  
item existed in a list of 100 items.**

© A+ Computer Science - www.apluscompsci.com

# **General Big O Chart for Searches**

Name	Best Case	Avg. Case	Worst Case
Linear/Sequential Search	O(1)	O(N)	O(N)
Binary Search	O(1)	O( $\log_2 N$ )	O( $\log_2 N$ )

All searches have a best case run time of O(1) if written properly. You have to look at the code to determine if the search has the ability to find the item and return immediately. If this case is present, the algorithm can have a best case of O(1).

**open  
binarysearch.java  
binarysearchtester.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **Sorts**

# **Quadratic ( N<sup>2</sup> )**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# The Selection Sort

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Selection Sort

The selection sort does not swap each time it finds elements out of position. Selection sort makes a complete pass while searching for the next item to swap. At the end of a pass once the item is located, one swap is made.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

## Selection Sort

```
void selectionSort( int[] ray )
{
    for(int i=0; i< ray.length-1; i++){
        int min = i;
        for(int j = i+1; j< ray.length; j++)
        {
            if(ray[j] < ray[min])
                min = j;          //find location of smallest
        }
        if( min != i) {
            int temp = ray[min];
            ray[min] = ray[i];
            ray[i] = temp;    //put smallest in pos i
        }
    }
}
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Selection sort is pretty effective for small lists, but pretty horrible is used on large lists.

Selection sort consists of two loops.

The outer loops run based on the number of items in the list.

The inner loop runs to find the items that need to be moved. The inner loop either locates the spot with the smallest value or the spot with the largest value. After the inner loop completes, a swap may occur if needed. At most, selection sort will make one swap per pass. A pass is one complete execution of the inner loop.

# Selection Sort

	0	1	2	3	4
<b>pass 0</b>	9	2	8	5	1
<b>pass 1</b>	1	2	8	5	9
<b>pass 2</b>	1	2	8	5	9
<b>pass 3</b>	1	2	5	8	9
<b>pass 4</b>	1	2	5	8	9

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

Selection sort is pretty effective for small lists, but pretty horrible is used on large lists.

Selection sort consists of two loops.

The outer loops run based on the number of items in the list.

The inner loop runs to find the items that need to be moved. The inner loop either locates the spot with the smallest value or the spot with the largest value. After the inner loop completes, a swap may occur if needed. At most, selection sort will make one swap per pass. A pass is one complete execution of the inner loop.

```
public void selSort(Comparable[] stuff){  
    for(int i=0;i<stuff.length-1;i++)  
    {  
        int spot=i;  
        for(int j=i;j<stuff.length;j++){  
            if(stuff[j].compareTo(stuff[spot])>0)  
                spot=j;  
        }  
        if(spot==i) continue;  
        Comparable save=stuff[i];  
        stuff[i]=stuff[spot];  
        stuff[spot]=save;  
    }  
}
```

How many swaps  
per pass?

Selection Sort  
W/Objects

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Selection Sort in Action

## Original List

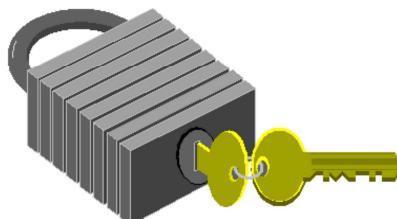
Integer[] ray = {90,40,20,30,10,67};

pass 1 - 90 40 20 30 10 67  
pass 2 - 90 67 20 30 10 40  
pass 3 - 90 67 40 30 10 20  
pass 4 - 90 67 40 30 10 20  
pass 5 - 90 67 40 30 20 10  
pass 6 - 90 67 40 30 20 10

**open  
selectionsort.java  
selectionsorttester.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# The Insertion Sort



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **Insertion Sort**

**The insertion sort first selects an item and moves items up or down based on the comparison to the selected item.**

**The idea is to get the selected item in proper position by shifting items around in the list.**

```
void insertionSort( int[] stuff)
{
    for (int i=1; i< stuff.length; ++i)
    {
        int val = stuff[i];
        int j=i;
        while(j>0&&val<stuff[j-1]) {
            stuff[j]=stuff[j-1];
            j--;
        }
        stuff[j]=val;
    }
}
```

## Insertion w/primitives

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

```
void insertionSort( Comparable[] stuff){
    for (int i=1; i< stuff.length; ++i){
        int bot=0, top=i-1;
        while (bot<=top){
            int mid=(bot+top)/2;
            if (stuff[mid].compareTo(stuff[ i ])<0)
                bot=mid+1;
            else top=mid-1;
        }
        Comparable temp= stuff[i];
        for (int j=i; j>bot; --j)
            stuff[ j]= stuff[ j-1];
        stuff[bot]=temp;
    }
}
```

## Insertion w/Objects

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

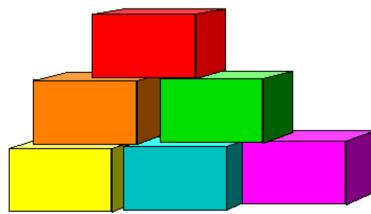
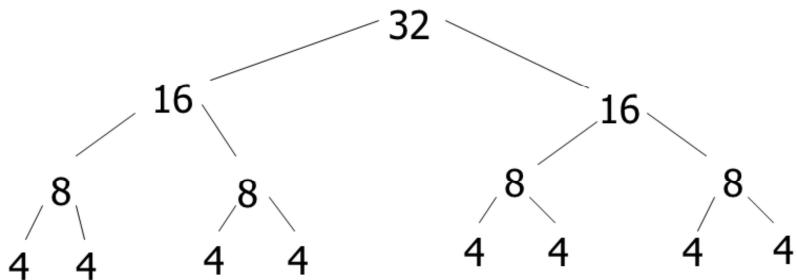
**open**  
**insertionsort.java**  
**insertionsorttester.java**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **Divide and Conquer Algorithms $O(N \log_2 N)$**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Divide and Conquer



© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# The QuickSort

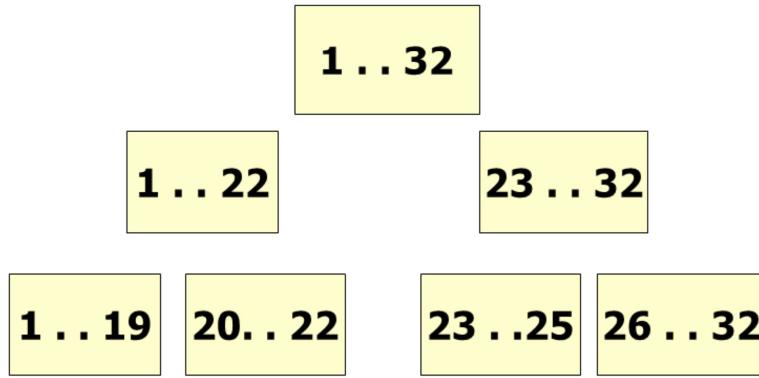
© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Quick Sort

Quick sort finds a pivot value. All numbers greater than the pivot move to the right and all numbers less move to the left.

This list is then chopped in two and the process above is repeated on the smaller sections.

# Quick Sort



Quick sort chops up the list into smaller pieces as to avoid processing the whole list at once.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# quickSort Algorithm

```
void quickSort(Comparable[] stuff, int low, int high)
{
    if (low < high)
    {
        int spot = partition(stuff, low, high);
        quickSort(stuff, low, spot);
        quickSort(stuff, spot+1, high);
    }
}
```

Arrays.sort( ) uses the quickSort  
if sorting primitives.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# partition Algorithm

```
int partition(Comparable[] stuff, int low, int high)
{
    Comparable pivot = stuff[low];
    int bot = low-1;
    int top = high+1;
    while(bot < top) {
        while (stuff[--top].compareTo(pivot) > 0);
        while (stuff[++bot].compareTo(pivot) < 0);
        if(bot >= top)
            return top;
        Comparable temp = stuff[bot];
        stuff[bot] = stuff[top];
        stuff[top] = temp;
    }
}
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Quick Sort in Action

## Original List

Integer[] ray = {90,40,20,30,10,67};

pass 1 - 67 40 20 30 10 90  
pass 2 - 10 40 20 30 67 90  
pass 3 - 10 40 20 30 67 90  
pass 4 - 10 30 20 40 67 90  
pass 5 - 10 20 30 40 67 90

**The quickSort has a  $N * \log_2 N$  BigO.**

## **quickSort**

**The quickSort method alone has a  $\log_2 N$  run time, but cannot be run without the partition method.**

## **Partition**

**The partition method alone has an  $N$  run time and can be run without the quickSort method.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

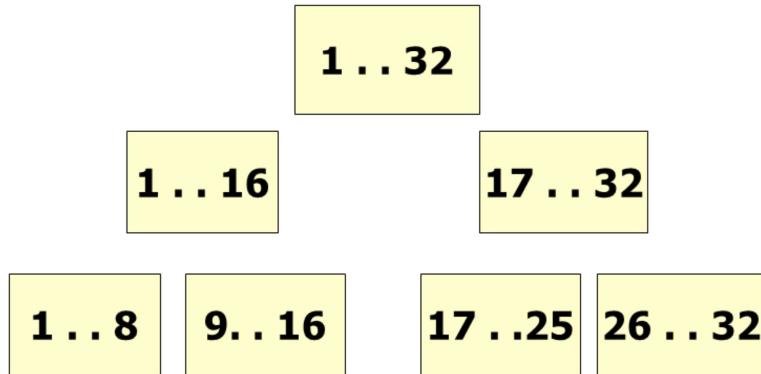


© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Merge Sort

**Merge sort splits the list into smaller sections working its way down to groups of two or one. Once the smallest groups are reached, the merge method is called to organize the smaller lists. Merge copies from the sub list to a temp array. The items are put in the temp array in sorted order.**

# Merge Sort



Merge sort chops in half repeatedly to avoid processing the whole list at once.

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **mergeSort Algorithm**

```
void mergeSort(Comparable[] stuff, int front, int back)
{
    int mid = (front+back)/2;
    if(mid==front) return;
    mergeSort(stuff, front, mid);
    mergeSort(stuff, mid, back);
    merge(stuff, front, back);
}
```

**Collections.sort( ) uses the mergeSort.**

**Arrays.sort( ) uses mergeSort for objects.**

```

void merge(Comparable[] stuff, int front, int back)
{
    Comparable[] temp = new Comparable[back-front];
    int i = front, j = (front+back)/2, k =0, mid =j;
    while( i<mid && j<back) {
        if(stuff[i].compareTo(stuff[j])<0)
            temp[k++]= stuff[i++];
        else
            temp[k++]= stuff[j++];
    }
    while(i<mid)
        temp[k++]= stuff[i++];
    while(j<back)
        temp[k++]= stuff[j++];
    for(i = 0; i<back-front; ++i)
        stuff[front+i]=temp[i];
}

```

## Merge W/Objects

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Merge Sort in Action

## Original List

Integer[] stuff = {90,40,20,30,10,67};

pass 0 - 90 20 40 30 67 10  
pass 1 - 20 40 90 30 67 10  
pass 2 - 20 40 90 30 10 67  
pass 3 - 20 40 90 10 30 67  
pass 4 - 10 20 30 40 67 90

**The mergeSort has a  $N * \log_2 N$  BigO.**

## **mergeSort**

**The mergeSort method alone has a  $\log_2 N$  run time, but cannot be run without the merge method.**

## **Merge**

**The merge method alone has an  $N$  run time and can be run without the mergeSort method.**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Speed

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Runtime Analysis

```
for( int i=0; i<20; i++)  
    System.out.println(i);
```

Which section of  
code would execute  
the fastest?

```
for( int j=0; j<20; j++)  
    for( int k=0; k<20; k++)  
        System.out.println(j*k);
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# Runtime Analysis

```
ArrayList<Integer> iRay;  
iRay = new ArrayList<Integer>();  
for( int i=0; i<20; i++)  
    iRay.add(i);
```

Which section of  
code would execute  
the fastest?

```
ArrayList<Double> dRay;  
dRay = new ArrayList<Double>();  
for( int j=0; j<20; j++)  
    dRay.add(0,j);
```

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)

# **General Big O Chart for N<sup>2</sup> Sorts**

Name	Best Case	Avg. Case	Worst
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$
Bubble Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$
Insertion Sort	$O(N)$ (@)	$O(N^2)$	$O(N^2)$

@ If the data is sorted, Insertion sort should only make one pass through the list. If this case is present, Insertion sort would have a best case of  $O(n)$ .

# **General Big O Chart for NLogN Sorts**

Name	Best Case	Avg. Case	Worst
------	-----------	-----------	-------

Merge Sort	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N \log_2 N)$
------------	-----------------	-----------------	-----------------

QuickSort	$O(N \log_2 N)$	$O(N \log_2 N)$	$O(N^2)$ (@)
-----------	-----------------	-----------------	--------------

@ QuickSort can degenerate to  $N^2$ . It typically will degenerate on sorted data if using a left or right pivot. Using a median pivot will help tremendously, but QuickSort can still degenerate on certain sets of data. The split position determines how QuickSort behaves.

# **Continue work on Lab22**

© A+ Computer Science - [www.apluscompsci.com](http://www.apluscompsci.com)