CS 112 – Introduction to Computing II

Wayne Snyder
Computer Science Department
Boston University

Today

- Conclusions on Iterative Sorting:
  - Complexity of Insertion Sort
- Recursive Sorting Methods and their Complexity:
  - Mergesort
- Conclusions on sorting algorithms and complexity

Next Time:

- Introduction to Linked Lists: Reference types on steroids!
- Iterative algorithms on Linked Lists
- (Reading: On the web site)

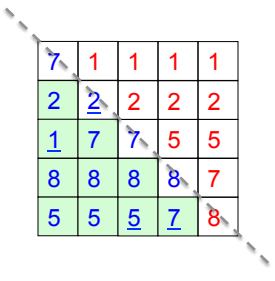
Computer Science

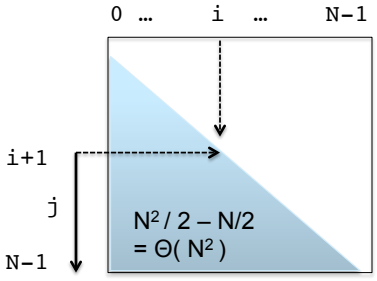
Conclusions on Complexity of Iterative Sorts

Computer Science

Recall why Selection sort (in all cases) is $\Theta(N^2)$:

```
for (int i = 0; i < N; i++) {
   for (int j = i+1; j < N; j++) {
            ….   Θ(1)   ….
   }
}
```





Sorting: Insertion Sort

Computer Science

Now let's look at Insertion Sort…..

Let's **count** the number of **calls to less(...)** in the worst case, which in fact is a reverse sorted list....

Observe that the outer loop runs N-1 times, and less is called

1 time, then 2 times, ... then (N-2) times, then finally (N-1) times.

```
public static void insertionSort(int[] a) {
  int N = a.length;
    for (int i = 1; i < N; i++) {
      for (int j = i; j > 0 && less(a[j], a[j-1]); j--) {
        swap(a, j, j-1);
      }
    }
}
```

3

Complexity of Insertion Sort

Computer Science

Now let's look at the diagram, coloring a slot blue if it was compared with the new key being inserted:

8	7	5	2	1
7	8	7	3	2
5	5	8	7	5
2	2	2	8	7
1	1	1	1	8

1 + 2 + 3 + 4 = 10 calls to less(...)

7	1	1	1	1
2	2	2	2	2
1	7	7	5	5
8	8	8	8	7
5	5	5	7	8

It is the same as for Selection Sort: $N^2/2 - N/2$ calls to less(....)

This is for a reverse sorted list! What about an already sorted list?

4

Complexity of Insertion Sort

Computer Science

For an already sorted list, Insertion Sort does something very smart: it just checks to see that each key is not less than the one above it, and doesn't go any further!

1	1	1	1	1
2	2	2	2	2
5	5	5	5	5
7	7	7	7	7
8	8	8	8	8

1 + 1 + 1 + 1 = 4 calls to less(...)

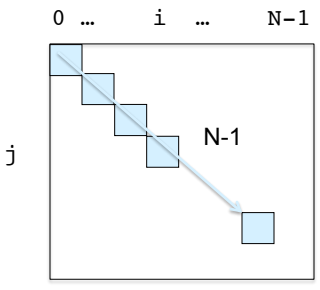
5

Conclusions on Complexity of Iterative Sorts

Computer Science

For the best case of Insertion Sort, we only do N-1 comparisons---one comparison per loop---to check that the given item is already in the correct place, so it is Θ(N)

1	1	1	1	1
2	2	2	2	2
5	5	5	5	5
7	7	7	7	7
8	8	8	8	8



Punchline: Insertion Sort adapts to its input, and can less work than Selection Sort, except in the worst case of a reverse-sorted list, where they both do the same!

6

Conclusions on Complexity of Iterative Sorts

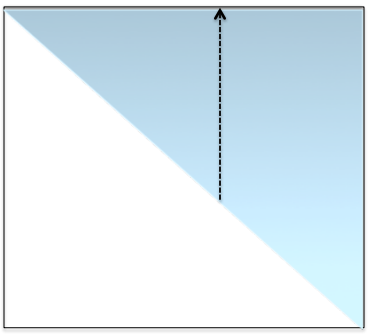
Computer Science

What about the Average Case?

For the worst case of Insertion Sort, observe that the worst thing that can happen is each number we insert is the smallest we have seen so far:

So at each step of the outer loop, the new item goes all the way up to the top:

8	7	5	2	1
7	8	7	3	2
5	5	8	7	5
2	2	2	8	7
1	1	1	1	8



$N^2 / 2 = \Theta(N^2)$

7

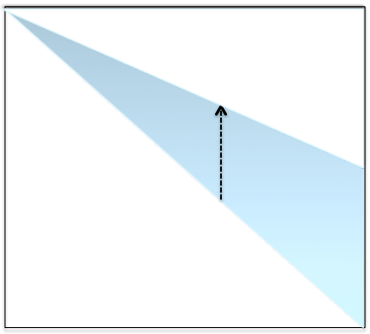
Conclusions on Complexity of Iterative Sorts

Computer Science

For the Average Case of Insertion Sort, observe that when we insert an *arbitrary* number into a ordered list, on average we go half way up:

```
insert  k into    13  10  9  7  6  4  3  2
```

So at each step of the outer loop, on average the new item goes half way up to the top:



$N^2 / 4 = \Theta(N^2)$

8

Iterative Sorting: Conclusions on Time Complexity

Computer Science

Algorithm	Worst-case Input	Worst-case Time	Best-case Input	Best-case Time	Average-case Input	Average-case Time
Selection Sort	Any!	$\Theta(N^2)$	Any!	$\Theta(N^2)$	Any!	$\Theta(N^2)$
Insertion Sort	Reverse Sorted List	$\Theta(N^2)$	Already Sorted List	$\Theta(N)$	Random List	$\Theta(N^2)$

Conclusions:

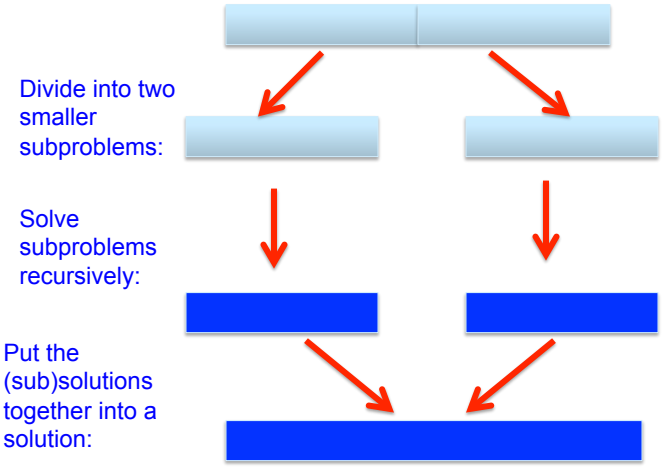
- **Selection Sort** is **inflexible** and does $\Theta(N^2)$ comparisons in all cases;
- **Insertion Sort** in the worst case does no better than Selection Sort, but **adapts to its input**: it performs better the "more sorted" the input it; in the case of an already sorted list, it simply checks that the list is sorted.

9

Recursive Sorting: Merge Sort

Computer Science

Merge Sort is a relative simple application of **recursive (Divide and Conquer) reasoning** to the problem of sorting:



10

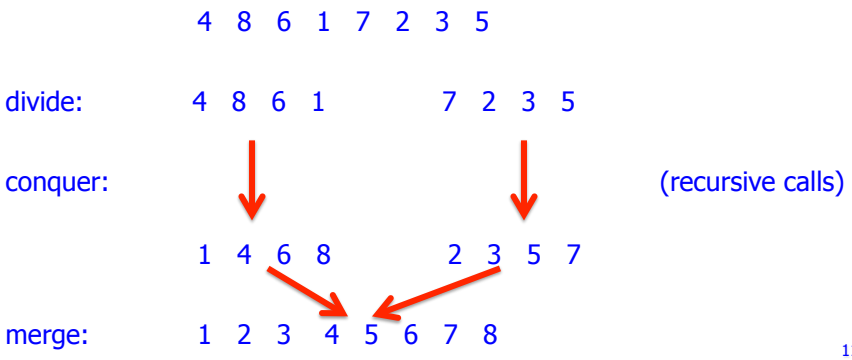
Merge Sort

Computer Science

Merge Sort is a relative simple application of **bottom up recursive (Divide and Conquer) reasoning** to the problem of sorting:

Base Case: If the list is empty or has only one element, stop;

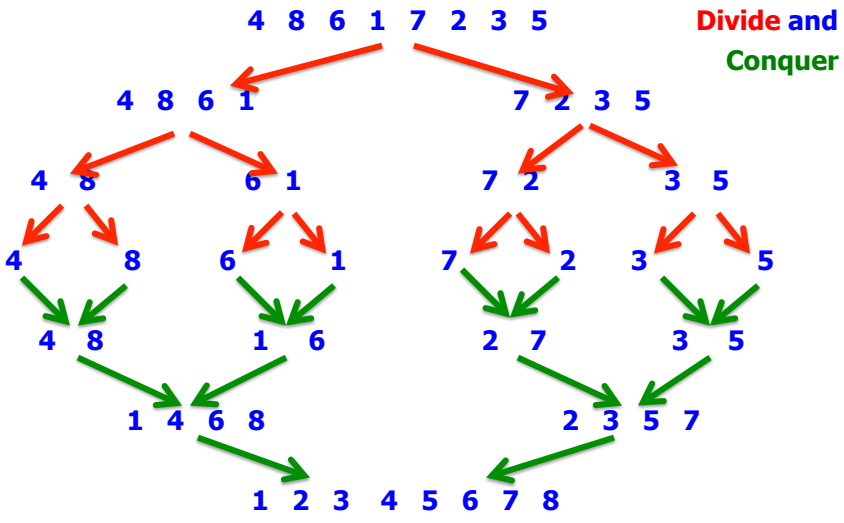
Recursive Case: If the list has 2 or more elements, divide in half (best you can), sort each separately, and then merge the two sorted lists into one sorted list:



11

Complexity of Merge Sort

Computer Science



12

Sorting: Merge Sort

Computer Science

```
private static void merge(Comparable [] a, Comparable [] aux, int lo, int mid, int hi) {
    // copy to aux[]
    for (int k = lo; k <= hi; k++) {
        aux[k] = a[k];
    }

    // merge back to a[]

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        if      (i > mid)              a[k] = aux[j++];      // left side exhausted
        else if (j > hi)               a[k] = aux[i++];      // right side exhaused
        else if (less(aux[j], aux[i])) a[k] = aux[j++];      // smallest on right side
        else                           a[k] = aux[i++];      // minimal on left side
    }
}

// mergesort a[lo..hi] using auxiliary array aux[lo..hi]

private static void mergeSort(Comparable [] a, Comparable [] aux, int lo, int hi) {
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    mergeSort(a, aux, lo, mid);
    mergeSort(a, aux, mid + 1, hi);
    merge(a, aux, lo, mid, hi);
}

private static void mergeSort(int[] a) {
    int[] aux = new int[a.length];
    mergeSort(a, aux, 0, a.length-1);
}
```


Complexity of Merge Sort

Computer Science

Let's **count** the number of **comparisons** (calls to less)

Observe that **less** is called in only one place, in **merge**, so we start by thinking about what happens when we merge two ordered lists.

What is the best thing that can happen when merging two ordered lists?

All the elements in one list are less than the elements in the other list, e.g., in an already ordered list:

1 2 3 4 5 6 7 8

How many comparisons?

Complexity of Merge Sort

Let's **count** the number of **comparisons** (calls to less)

Observe that **less** is called in only one place, in **merge**, so we start by thinking about what happens when we merge two ordered lists.

What is the best thing that can happen when merging two ordered lists?

All the elements in one list are less than the elements in the other list, e.g., in an already ordered list:

1 2 3 4 5 6 7 8

How many comparisons? 4 (in general: Θ(N) for N elements)

15

Complexity of Merge Sort

What is the WORST thing that can happen when merging two ordered lists?

The rightmost elements in each list are the two largest elements: and the last comparison must compare these two:

1 2 3 7 5 6 4 8

So every element except for the largest has to be compared before moving down.

How many comparisons?

16

Complexity of Merge Sort

What is the WORST thing that can happen when merging two ordered lists?

The rightmost elements in each list are the two largest elements: and the last comparison must compare these two:

1 2 3 7 5 6 4 8

So every element except for the largest has to be compared before moving down.
How many comparisons? 7 (in general: N-1 or Θ(N) for N elements)

Conclusion: Merge takes linear time, Θ(N), in the number of comparisons.

Check: Suppose we count the number of moves (assignments)? Then obviously each element is moved once per merge, giving N = Θ(N) moves.

Punchline: Merge takes linear time: Θ(N).

17

Complexity of Merge Sort

Now: how many times is Merge called?

Or:

How many times can you divide a list of size N in half when you Divide before Conquering?

N N/ 2 N/4 ..... 4 2 1

Let's write it the other way:

1 2 4 ... N

Or:

2^0 2^1 2^2 ..... $2^{\log(N)} = N$

Answer: **Θ ($\mathbf{Log_2}$ N)**

18

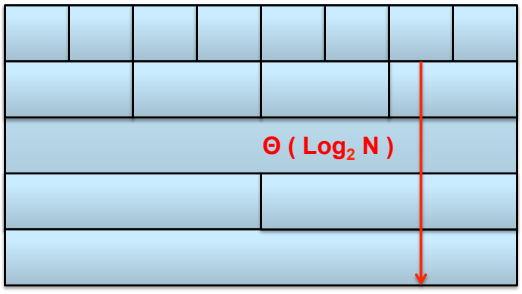
Complexity of Merge Sort

Computer Science

So: Merge takes Θ(N) comparisons in all cases.

You can divide the list of size N in half Θ (Log_2 N) times.

Punchline: Mergesort takes Θ (N * Log_2 N) comparisons in all cases:



Subproblem size	Number of Subproblems	Number of Comparisons
(Initial Problem)		
2	N/2	Θ(N/2 * 2) = Θ(N)
N/2	2	Θ(2 * N/2) = Θ(N)
N	1	Θ(1 * N) = Θ(N)

19

Complexity of Merge Sort

Computer Science

BUT there is another issue other than Time Complexity here:

How much memory does Merge Sort require?

Note that we have to have ANOTHER ARRAY just as big to do the merge step:

So Merge Sort is faster than the iterative sorts in the worst case, but **requires twice as much storage.** This may be a factor in large arrays!

20

Sorting: Conclusions on Time Complexity

Computer Science

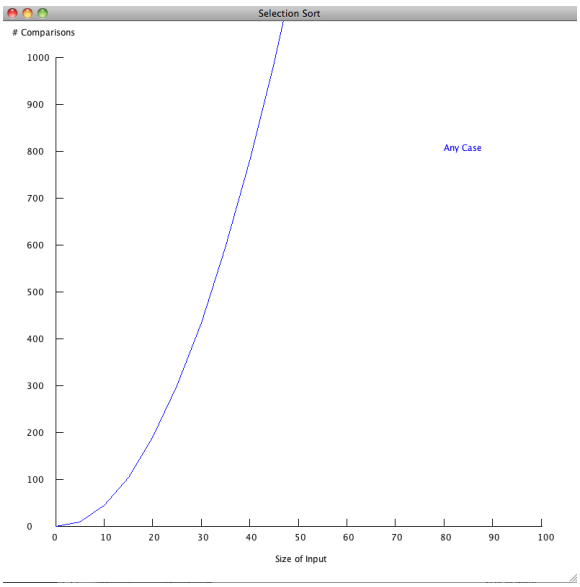
Algorithm	Worst-case Input	Worst-case Time	Best-case Input	Best-case Time	Average-case Input	Average-case Time
Selection Sort	Any!	$\Theta(N^2)$	Any!	$\Theta(N^2)$	Any!	$\Theta(N^2)$
Insertion Sort	Reverse Sorted List	$\Theta(N^2)$	Already Sorted List	$\Theta(N)$	Random List	$\Theta(N^2)$
Mergesort	*Complicated!*	$\Theta(N^*\log(N))$	Already Sorted List	$\Theta(N^*\log(N))$	Random List	$\Theta(N^*\log(N))$

Memory Usage: Selection Sort and Insertion Sort can be done "in place" in the same array; Merge Sort requires an extra array just as big!

21

Comparing Sorting Algorithms

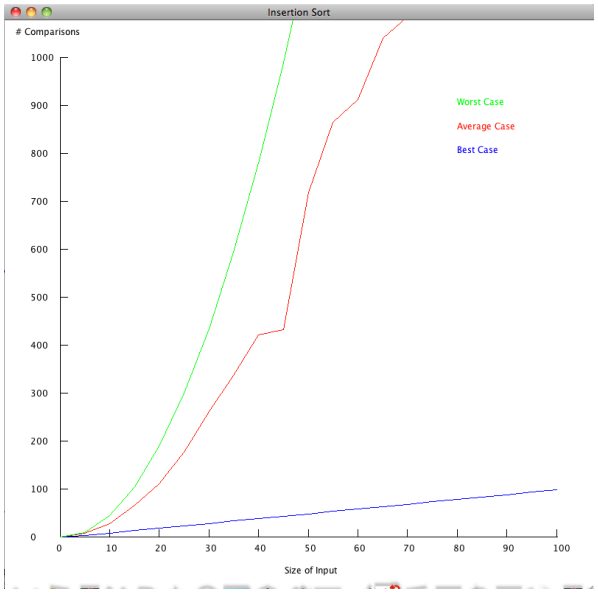
Computer Science



22

Comparing Sorting Algorithms

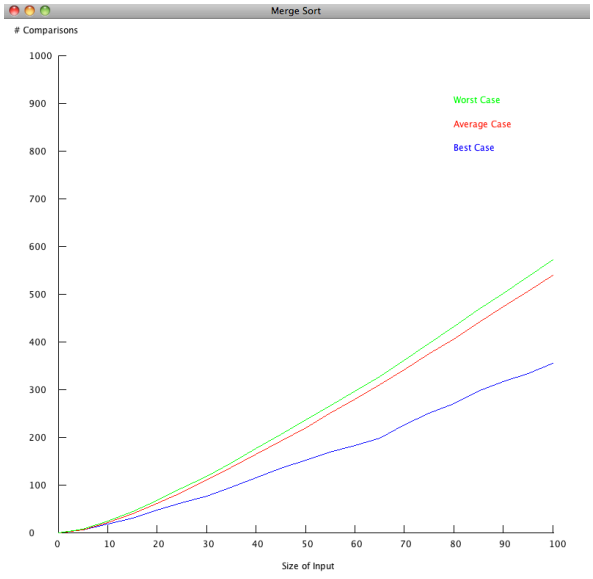
Computer Science



23

Comparing Sorting Algorithms

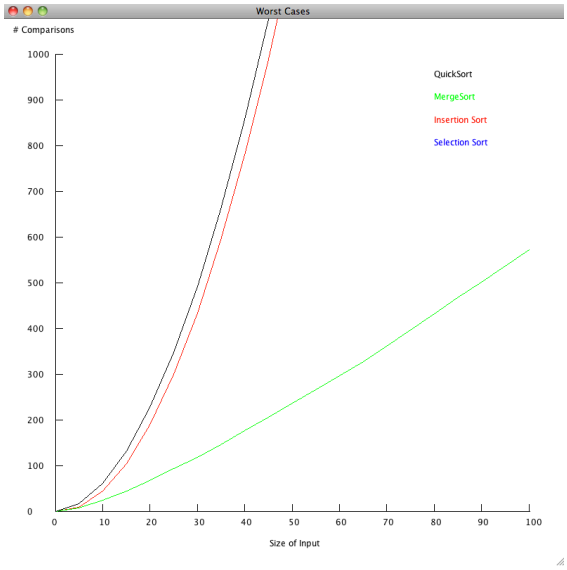
Computer Science



24

Comparing Sorting Algorithms

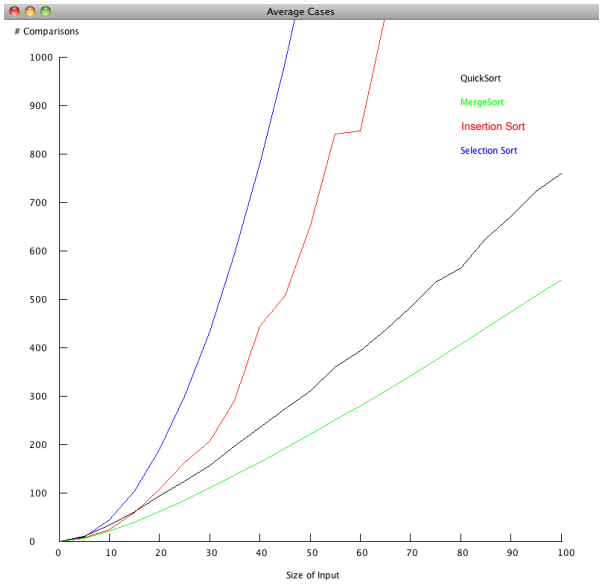
Computer Science



25

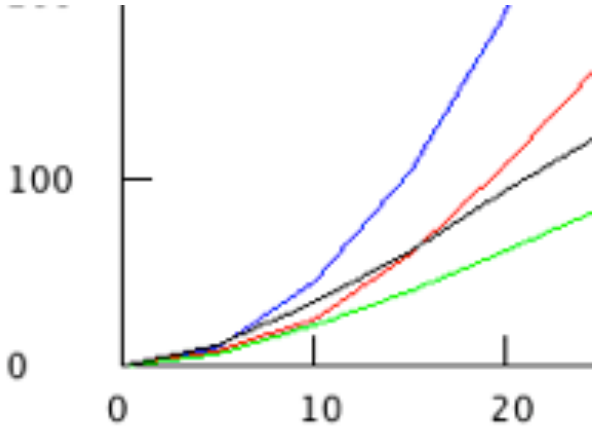
Comparing Sorting Algorithms

Computer Science



26

Comparison of Sorting Algorithms: Average Case



27

Timing Java Code

But is counting comparisons the best way to analyze algorithms? What about how much TIME they take??

This turns out to be a complicated question, because the actual time depends on many, many factors:

- How fast is your processor? Do you have more than 1 processor?
- How many other processes are running? (Example: the Java garbage collector!)
- How much memory do you have? Does this affect really big inputs?
- What operating system?
- Etc., etc., etc.

To do this right, you have to specify ALL these parameters, and run a standard platform with standard benchmarks; this is in fact done when testing new processors.

But assuming we are running two different algorithms on the same platform, we should be able to get some interesting results. Let's think about how to time Java code…..

28

Timing Java Code

Here is a simple way to time a region of Java code:

```
long startTime = System.currentTimeMillis();

// some code you want to time

long endTime = System.currentTimeMillis();

System.out.println("Total execution time: " + (endTime – startTime));
```

To get more precision, you can do the code 100000 times, then divide by 100000, etc.

29

Timing Java Code

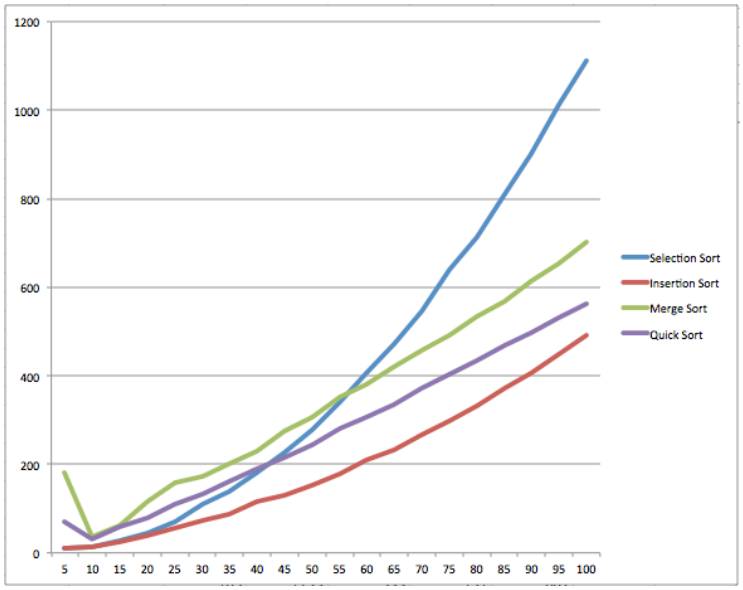
Here is a sample of what I wrote to time our sorting algorithms:

```
for(int i = 5; i <= 200 ; i+=5){
   int[][] a = new int[100000][0];
   for(int j = 0; j < 100000; ++j)
      a[j] = genRandomArray(i);
   long startTime = System.currentTimeMillis();

   for(int j = 0; j < 100000; ++j)

      selectionSort(a[j]);         // code to be timed goes here

   long endTime = System.currentTimeMillis();
   System.out.println(i + " " + (endTime - startTime));
}
```

30

Timing Java Code: Average Case for Actual Time

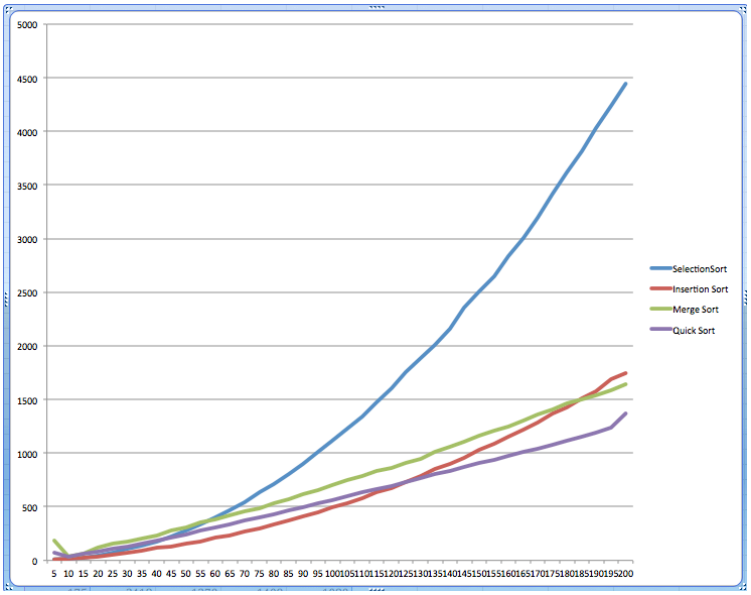
Computer Science



31

Timing Java Code: Average Case for Actual Time

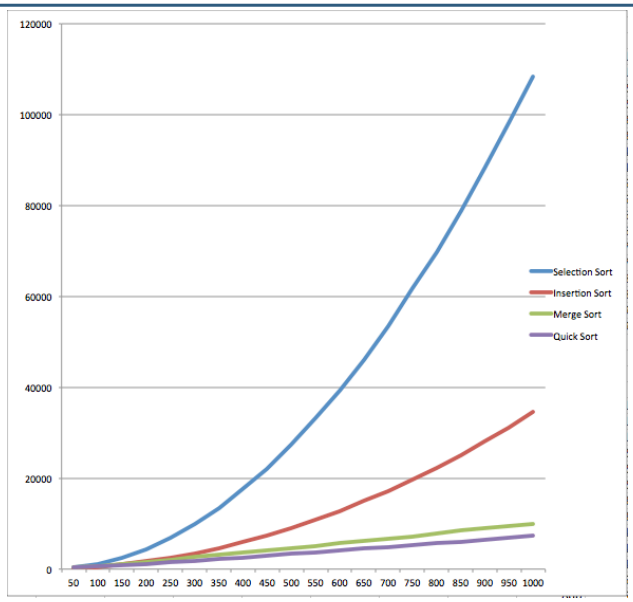
Computer Science



32

Timing Java Code

Computer Science



33