Report

Algorithm effectiveness

-StringBuilder VS + and insertion sort VS merge sort

Author: Rasmus Skedinger
Examiner: Jonas LUNDERG
Semester: VT 2016
Subject: Computer Science

Contents

1 Process

In this section a description of how the experiment process was created to give the results needed. It will describe two different methods in constructing the algorithms and the thought process behind them. One method turned out to be not that great memory wise but gave the results needed to proceed so it was decided to keep it, and a better one that i should have used from the beginning.

1.1 StringBuilder and +

The only way to do this is to find out how many times it can run the function during one second since there is no end of this test.To measure these methods. The time needs to be taken as close to the measured part of the code as possible and then see how many times it could iterate over the method during that one second.

```
while(time < 1000000000){
  before = System.nanoTime();
  theString = theString +longStr;;
  time = time+(System.nanoTime()- before);
  times++;
}
```

And the same for Stringbuilder:

```
while(time < 1000000000){
  before = System.nanoTime();
  sb.append(longStr);
  time = time+(System.nanoTime()- before);
  times++;
}
```

This runs ten times and an median value is calculated and printed to the user for easy reading. Then minimised the runtime in the while loop to one millisecond instead of a second because it ended up taking 12GB of ram though.

1.2 Insertion sort and Merge sort

In insertion sort and merge sort there was the limit of the size of the array that was going to be sorted. So the time is simply taken before and after the sorting is done.

```
before = System.nanoTime();
sa.mergeSort(sArr);
after = System.nanoTime() - before;
```

And again pick the median time from running it ten times.

2 The results are in!

When the results on the resource analysis came there are some big differences in both the StringBuilder and +, and also between the sorting algorithms themselves. Between the sorting algorithms is also a big difference between an int array and a String.

2.1 StringBuilder and +

Function	Test type	Run times
StringBuilder	Letter	40499730
	Sentence	8029639
String +	Letter	69316
	Sentence	4485

Table 2.1: Results table of String builder and +.

As table 2.1 shows StringBuilder works faster in both the sentence and in a single letter test. The reason for it being faster is that string + is an object that is not mutable, it takes all the added objects and return them to a new unchangeable object. What string builder is, is that it is an mutable object that accepts changes easier. Therefore with StringBuilder this means the computer has to work with lesser objects then with String.

2.2 Insertion sort and Merge sort

Algorithm	Test type	Runtime Time
Insertion sort	`int[10000]`	15919152
Merge sort	`int[10000]`	1641936
Insertion sort	`String[10000]`	113657457
Merge sort	`String[10000]`	1994800

Table 2.2: Results table of the sorting algorithms.

In insertion sort and merge sort there are two differences. One between the algorithms themselves and second between the algorithms with the string against int. Let's first take a look at the differences between integer and string. When the compeering is done between each element in the array's, with an integer array it is easy to deal with compeering two integer values against one another and sort for the highest one but with String, you have to convert the integer to it's corresponding int value in the ascii table to compare them and sort them, this is done with the extra `compareTo` function which must run. And that takes extra time. The second big difference is between the two algorithm themselves. The insertion sort algorithm goes through the whole array and scans for the lowest value and puts it first, this is done time for each element in the array and that is very time consuming. In merge sort it builds a tree down to each element and than sorts the elements on the way back up to it's full length again, the final result is a lot fewer steps than the insertion sort and that is the reason why it is faster.