Assignment 1 Data Structures and Algorithms

Linus Åberg
Hjalmar Thunberg

2020-02-21

Complexity Analysis

Question 1:

a.

$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1}{6} = \frac{n(2n^2+n+2n+1)}{6} = \frac{2n^3+3n^2+n}{6}$. This gives us the complexity $O(n^3)$.

b.

$\sum_{i=1}^{\log n} 2^i = \frac{2^{\log n}-1}{2-1} = n * 2 - 1 = 2n - 1$ which gives us the complexity $O(n)$

c.

$2x^2 + 16x^3 = O(N^3)$

d.

$(x - \log x^3)(x - 2\sqrt{x}) + 4x \log x^2 = x^2 - x \log x^3 - 2x\sqrt{x} + (2\sqrt{x} * \log x^3) + 4x \log x^2$

We have the terms x^2, $x \log x$, $x\sqrt{x}$, $\sqrt{x} * \log x$

Since $\sqrt{x} < x$ and $\log x < x$ we know that we can exclude those terms. We also know that $x^2 > x$ and $x \log x > x$ so the complexity must be one of these two. And since $x^2 > x \log x$ the complexity is $O(n^2)$

Question 2:

Show that $4^n \notin O(2^n)$

$C * 4^n <= 2^n = C <= \frac{2^n}{4^n}$

there exists no positive integer that satisfies $C <= \frac{2^n}{4^n}$

Question 3:

```
int a[] = {a0, a1, a2, a3, ... , an-1, an}
int sum = a0
for (int i = 1 ; i < a.length ; i++)
 sum += ai * x ^ i;
```

Recursion

Question 4:

See contained file Question4.java for code implementation. The complexity of the code is $O(n)$.

Question 5:

Solve the complexity of the following recurrences using the Master Method:

- $T(n) = 4T(\frac{n}{2}) + \theta(n^2)$

  $A = 4, B = 2, k = 2, B^k = 4$. If we follow the master theorem we know that $A = B^k$ which gives us $O(N^2 \log N)$.

- $T(n) = 5T(\frac{n}{2}) + \theta(n^2)$

  $A = 5, B = 2, k = 2, B^k = 4$. If we apply the master theorem we know that $A > B^k$ which gives us $O(N^{\log_2 5})$

Question 6:

See contained file Assignment1.java

Sorting Algorithms

Question 7:

See contained file Assignment1.java

Question 8:

Table 1: Summary of benchmarking study

BENCHMARK SUMMARY		**Execution Time (CPU seconds)**	
		sorted (ascending order)	**sorted (descending order)**
bubble sort	**array size: 100**	0.011856ms	0.00871ms
	array size: 100000	2.450648ms	3.560365ms
merge sort	**array size: 100**	0.069237ms	0.096516ms
	array size: 100000	18.074579ms	15.139832ms
insertion sort	**array size: 100**	0.017233ms	0.010389ms
	array size: 100000	2.459731ms	3.709379ms
quicksort (pivot: median)	**array size: 100**	0.037518ms	0.029105
	array size: 100000	29.750264ms	26.988493ms
quicksort (pivot: A[0])	**array size: 100**	0.10272ms	0.088564ms
	array size: 100000	1571.87171ms	1558.270053ms

In order to solve the question we created two arrays of size 100 which where pre-sorted in ascending and descending order [1-100] [100-1]. We also created two arrays of size 100 000 which where pre-sorted in ascending and descending order [1-100 000] and [100 000-1]. We then run our sorting algorithms on these arrays.

Both of the quicksort algorithms performed bad for arrays of size 100 000 elements, especially the one for A[0] where we got around 1500 ms for both ascending and descending order.

It was a very small difference between the two guicksort algorithms for arrays sorted in ascending order and of size 100. The difference is approximately 0.1 ms. For array size 100 000 we have a bigger difference. We go from 29.750264 ms to 1571.87171 ms.

The difference between the two quicksort algorithms for arrays sorted in descending order of size 100 it is a very small difference. The difference is approximately 0.06 ms. For array size 100 000 we have a bigger difference. We go from 26.988493 ms to 1558.270053 ms.

The insertion sort for arrays sorted in descending order of size 100 compares very good with the rest of the sorting algorithms where it is beaten by the quicksort algorithm which uses median as pivot and the bubblesort. In descending order it comes in third place with bubblesort and quciksort performing just a bit better. For array size 100 000 ascending order it performed very good with 2.45973 ms and is only beaten by the bubblesort which run in 2.450648 ms. In descending order it has a run time of 3.709379 ms which is only beaten by the bubblesort which have a run time of 3.560365 ms.

There is no big difference between the mergesort for the different cases.