Algorithms Homework 4

Aladdin Persson (aladdin.persson@hotmail.com)

1 Homework 4

1.1 Question 1

I haven't chosen to do all of the reccurence relations, mainly because the method I used was essentially the same for all of them. Rather I'd like for you to check and give me feedback on just a few of them.

a)

Let's assume the base case is $T(1)$ which equals some constant term.

$$\begin{aligned}
&T(n) = T(n-1) + cn = \\
&T(n-2) + c(n-1) + cn = \\
&T(n-3) + c(n-2) + c(n-1) + cn = ... = \\
&T(n-k) + c(n-k+1) + c(n-k+2) + ... + cn
\end{aligned} \tag{1}$$

With $k = n-1$ we obtain

$$\begin{aligned}
&T(1) + c\sum_{j=0}^{k-1}(n-j) = \\
&T(1) + \sum_{j=0}^{k-1} cn - \sum_{j=0}^{k-1} cj = \\
&T(1) + \sum_{j=0}^{n-2} cn - \sum_{j=0}^{n-2} cj \leq \\
&T(1) + \sum_{j=0}^{n-2} cn = T(1) + (n-1)cn \in \mathcal{O}(n^2)
\end{aligned} \tag{2}$$

f) Let us assume that $n = 2^k$ and that the logarithm used is in base 2 which would be a lower bound since $log_2(x) \leq log_3(x)$, etc.

$$\begin{aligned}
&T(n) = 2T(\frac{n}{2}) + nlogn = \\
&T(n) = 2^2T(\frac{n}{2^2}) + nlog(\frac{n}{2}) + nlogn = \\
&T(n) = ... = 2^kT(\frac{n}{2^k}) + n(log(\frac{n}{2^{k-1}}) + log(\frac{n}{2^{k-2}} + ... + log(n)))
\end{aligned} \tag{3}$$

Let's remember that $k = log_2(n)$. And we also use the assumption that the above logarithms are in base 2. Following from Eq. (3) we obtain

$$\begin{aligned}
&T(n) = nT(1) + n((k-(k-1) + 2 + ... + k) = \\
&=nT(1) + n0.5(k+1)(k) = nT(1) + n(0.5(log_2(n)+1)log_2(n) \in \mathcal{O}(nlog^2(n))
\end{aligned}$$

1.2 Question 2

1, a) Since we are each iteration merging two arrays then there will be one array less for each iteration. Each time we merge we will do a constant times n operations. The reccurence we use is then $T(n,k) = T(k-1) + cn$

1,b) From the explanation in a) I think it should be clear that we need to do $\mathcal{O}(kn)$, from unrolling a total of k times and each time we do cn operations.

2,a) MultiMerge based on divide and conquer approach would work by first merging arrays 1,2: $1' = Merge(1,2)$ and arrays 3,4: $2' = Merge(3,4)$, etc. Then we do the same on 1', 2', etc.

2,b) Let us define A to be an array of all sorted arrays, where first element of A is the first sorted array etc. Also note that I am using 0-indexing in the pseudocode below.

Algorithm 1 MultiMerge(A)

```
if length of A is 1 then return A[0]
Initialize new_A as empty array
for idx = 1 to length of A with a step of 2 do
    new_A.append(merge(A[idx], A[idx-1]))
if modulus 2 of length(A)= 1 then new_A.append(last element of A)
return MultiMerge(new_A)
```

We can prove it by induction. Assume it holds if $p = 1$ which is trivial. Assume that MultiMerge is correct when considering $p-1$ arrays and then we wish to prove it holds for p arrays. If it holds for $p-1$ then it will merge these and let's call this result A, we then want to prove that merging A with another array B is correct. Here we are simply merging two arrays, which runs the merge function that we assume to be correct, hence it holds for p arrays.

2,c) The runtime function should be $T(n,k) = T(\frac{k}{2}) + cn\frac{k}{2}$