csci-5454-hw2

Jake Krol

September 2024

1 Q1

1.1 Q1a

The maximum prefix sum (**pref**) of the original array can either be the max prefix from the left array or the sum of the left array ($\mathbf{s}_l$) plus the prefix of the right array. Likewise, max suffix sum (**suff**) can either be the max suffix from the right array or the sum of the right ($\mathbf{s}_r$) array plus the suffix of the left array.

$$\mathbf{pref} = max(\mathbf{pref}_l, \mathbf{s}_l + \mathbf{pref}_r) \tag{1}$$

$$\mathbf{suff} = max(\mathbf{suff}_r, \mathbf{s}_r + \mathbf{suff}_l) \tag{2}$$

1.2 Q1b

Algorithm 1 max-subarray(A, low, high)

```
if high == low then
    return (low, low, low, low)                          ▷ base case
else
    mid ← ⌊(low+high)/2⌋
    (s_l, pref_l, suff_l, m_l) ← max-subarray(A, low, mid)
    (s_r, pref_r, suff_r, m_r) ← max-subarray(A, mid + 1, high)
    pref ← max(pref_l, s_l + pref_r)
    suff ← max(suff_r, s_r + suff_l)
    s_A ← s_l + s_r
    m_A ← max(m_l, m_r, suff_l + pref_r)
    return (s_A, pref, suff, m_A)
end if
```

Recursion splits arrays in half (by self-calling max-subarray) until the base case of size 1 is reached. As values start to return from max-subarray calls in the **else** block, the combination step computes max-prefix (pref), max-suffix (suff), total sum (s_A), and max subarray at the current step (m_A). This continues until the 2nd level of the recursion tree (now traveling upward) resolves and is combined into final results of pref, suff, s_A, and m_A .

2 Q2

2.1 Q2a

If squish, s, is already computed from $r1$ to $r2$, then the squish from $r1$ to $r2+1$ is just the element wise sum of the previous squish and the column $r2+1$.

$$s = \mathsf{squish}(A, r_1, r_2) \tag{3}$$

$$\mathsf{squish}(A, r_1, r_2 + 1) = s + A[0 \dots n-1, r_2 + 1] \tag{4}$$

Where $A[0 \dots n-1, r_2+1]$ is the column vector of the matrix at $r_2 + 1$.

2.2 Q2b

Algorithm 2 max-subarray(A)

```
s_current ← 0
s_max ← −∞
n ← LENGTH(A)
for i = 0 upto n − 1 do
    s_current ← max(0, s_current + A[i])
    s_max ← max(s_max, s_current)
end for
return s_max
```

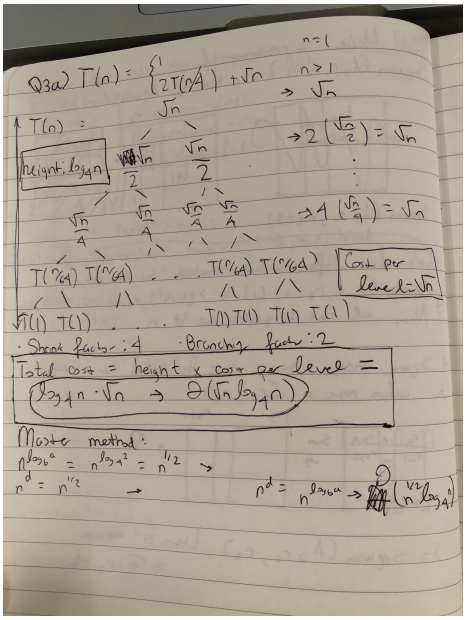
Algorithm 3 max-submatrix(A)

```
n ← LENGTH(A)
m_A ← −∞
for r_1 ← 0 upto n − 1 do                          ▷ left column idx
    sq ← ARRAY(n)                                  ▷ init array of length n
    for r_2 ← r1 upto n − 1 do      ▷ r_2 ≥ r_1    ▷ right column idx
        for i ← 0 upto n − 1 do                    ▷ row idx
            sq[i] ← sq[i] + A[i, r2]               ▷ squish columns for Kadane
        end for
        m_A ← max-subarray(sq)
    end for
end for
return m_A
```

Max-subarray uses Kadane's algorithm to iteratively track and test sums in $O(n)$. For a **max-submatrix**, we iterate through pairs of column start and stop indices, r_1, r_2. For each r_1,r_2 pair the subarray is squished by column vector additions $O(n^2)$, and the squished column vector is fed into max-subarray to compute the max-submatrix sum since each index in the submatrix is a possible sum with r_1, r_2 column start,stop indices over the rows represented as elements, i, of the squished array. The total cost is $O(n^3)$.

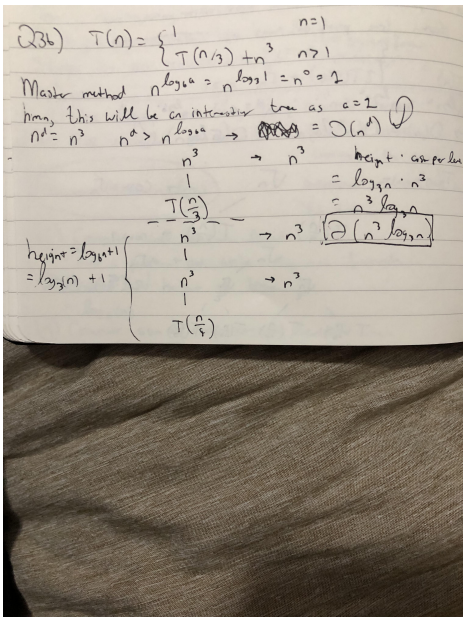
3 Q3

3.1 Q3a



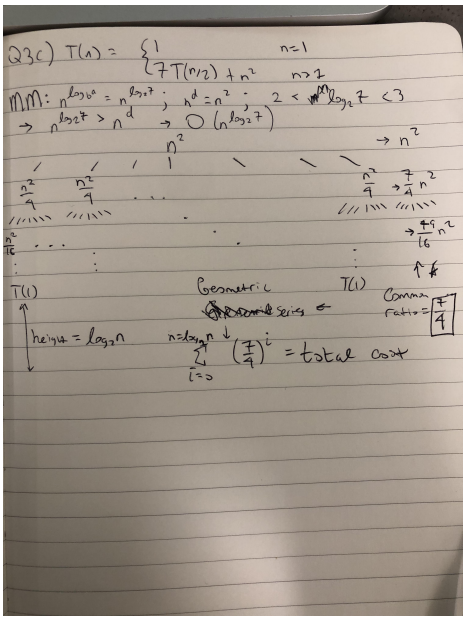
Recurrence $T(n) = 2T(\frac{n}{4}) + \sqrt{n}$ implies that problem size recursively shrinks by a factor of 4, and the cost of combining is $\sqrt{n}$. The recursion tree has a height of $\log_4(n) + 1$ and the cost of each level in the tree is $\sqrt{n}$. Multiplying the number of levels (height) by the cost at each level gives a total cost asymptotic to $\Theta(\sqrt{n} \log_4 n)$.

3.2 Q3b



The recursion tree for $T(n) = 2T(\frac{n}{3}) + n^3$ has a height of $\log_3(n) + 1$. The cost at each level is n^3. The total cost is $\Theta(n^3 \log_3 n)$.

3.3 Q3c



The recursion tree for $T(n) = 7T(n/2) + n^2$ has a shrink factor of 2, and a combination cost of n^2. The cost of the first level is n^2 while the cost at the ith level is $(\frac{7}{4})^i n^2$. The total cost is a function of the level i and can be written as a geometric sum:

$$\sum_{i=0}^{n=\log_2(n)} (\frac{7}{4})^i n^2 = \tag{5}$$

$$n^2 \sum_{i=0}^{n=\log_2(n)} (\frac{7}{4})^i = \tag{6}$$

$$\frac{7}{4}(\frac{1-(\frac{7}{4})^{\log_2(n)+1}}{1-\frac{7}{4}}) = \tag{7}$$

$$7(\frac{1-(\frac{7}{4})^{\log_2(n)+1}}{-3}) = \tag{8}$$

$$-\frac{7}{3}(1-(\frac{7}{4})^{\log_2(n)+1}) = \tag{9}$$

$$-\frac{7}{3}(1-(\frac{7}{4})^{\log_2 n} * (\frac{7}{4})) = \tag{10}$$

$$-\frac{7}{3}(1-\frac{7}{4}n^{\log_2 \frac{7}{4}}) \tag{11}$$

The total cost is $\Theta(n^{\log_2(\frac{7}{4})})$.