Problem Set 1

Justin Ely

Foundations of Algorithms
June 06, 2016
612-240-0924

1)

We wish to prove by induction that $\sum_{i=1}^{n} i^3 = (\frac{n(n+1)}{2})^2$. We start this by proving a trivial case of n=1.

$$1 = (\frac{1(1+1)}{2})^2 \tag{1}$$

Now we prove that the solution works for n+1.

$$\begin{align}
\sum_{i=1}^{k+1} i^3 &= \sum_{i=1}^{k} i^3 + (k+1)^3 \tag{2}\\
&= (\frac{k(k+1)}{2})^2 + (k+1)^3 \tag{3}\\
&= \frac{k^2(k+1)^2}{2^2} + \frac{2^2(k+1)^3}{2^2} \tag{4}\\
&= \frac{k^2(k+1)^2 + 2^2(k+1)^3}{2^2} \tag{5}\\
&= k^2(k^2+2k+1) + 2^2(k^2+2k+1)(k+1) \tag{6}\\
&= k^4 + 2k^3 + k^2 + (4k^2+8k+4)(k+1) \tag{7}\\
&= k^4 + 2k^3 + k^2 + 4k^3 + 12k^2 + 12k + 4 \tag{8}\\
&= k^4 + 6k^3 + 13k^2 + 12k + 4 \tag{9}\\
&= \frac{(k+1)^2(k+2)^2}{2^2} \tag{10}\\
&= (\frac{(k+1)(k+1+1)}{2})^2 \tag{11}
\end{align}$$

Which proves the induction hypothesis.

Using:

$$\begin{align}
(k+1)(k+1) &= k^2 + 2k + 1 \tag{12}\\
(k+2)(k+2) &= k^2 + 4k + 4 \tag{13}\\
(k+1)^2(k+2)^2 &= k^4 + 6k^3 + 13k^2 + 12k + 4 \tag{14}
\end{align}$$

2a)

```
[40, 17, 45, 82, 62, 32, 30, 44, 93, 10]
[17, 40, 45, 82, 62, 32, 30, 44, 93, 10]
[17, 40, 45, 82, 62, 32, 30, 44, 93, 10]
[17, 40, 45, 82, 62, 32, 30, 44, 93, 10]
[17, 40, 45, 62, 82, 32, 30, 44, 93, 10]
[17, 32, 40, 45, 62, 82, 30, 44, 93, 10]
[17, 30, 32, 40, 45, 62, 82, 44, 93, 10]
[17, 30, 32, 40, 44, 45, 62, 82, 93, 10]
[17, 30, 32, 40, 44, 45, 62, 82, 93, 10]
[10, 17, 30, 32, 40, 44, 45, 62, 82, 93]
```

2b)

```
Split:  [75, 56, 85, 90, 49, 26, 12, 48, 40, 47]  ->
        [75, 56, 85, 90, 49] & [26, 12, 48, 40, 47]
Split:  [75, 56, 85, 90, 49]  ->
        [75, 56] & [85, 90, 49]
Split:  [75, 56]  ->
        [75] & [56]
Merge: [75] with [56] -> [56, 75]
Split:  [85, 90, 49]  ->
        [85] & [90, 49]
Split:  [90, 49]  ->
        [90] & [49]
Merge: [90] with [49] -> [49, 90]
Merge: [85] with [49, 90] -> [49, 85, 90]
Merge: [56, 75] with [49, 85, 90] -> [49, 56, 75, 85, 90]
Split:  [26, 12, 48, 40, 47]  ->
        [26, 12] & [48, 40, 47]
Split:  [26, 12]  ->
        [26] & [12]
Merge: [26] with [12] -> [12, 26]
Split:  [48, 40, 47]  ->
        [48] & [40, 47]
Split:  [40, 47]  ->
        [40] & [47]
Merge: [40] with [47] -> [40, 47]
Merge: [48] with [40, 47] -> [40, 47, 48]
Merge: [12, 26] with [40, 47, 48] -> [12, 26, 40, 47, 48]
Merge: [49, 56, 75, 85, 90] with [12, 26, 40, 47, 48] ->
       [12, 26, 40, 47, 48, 49, 56, 75, 85, 90]
```

2c)

With $\frac{n}{k}$ sublists, each of length k elements, and insertion sort running at order $\Theta(n^2)$; then $\frac{n}{k}k^2 = nk$ to sort all lists.

2d)

Each merge operation takes on order n, operating $log(\frac{n}{k})$ times, thus merging all the sublists takes: $\Theta(nlog(\frac{n}{k}))$.

2e)

The largest value of k will come when the modified merge sort is of the same order as the standard merge sort, which happens when:

$$\Theta(nlog(n)) = \Theta(nk + nlog\frac{n}{k}) \tag{15}$$

We know that k cannot be a higher order than $log(n)$, as this would cause the modified algorithm to have a higher order worst-case growth rate than the standard merge sort. We can thus make the assumption that $k = log(n)$ is the valid solution, and test.

$$\begin{aligned}
\Theta(nlog(n)) &= \Theta(nk + nlog(\frac{n}{k})) && (16)\\
&= \Theta(nk + nlog(n) - nlog(k)) && (17)\\
&= \Theta(nlog(n) + nlog(n) - nlog(log(n))) && (18)\\
&= \Theta(2nlog(n) - nlog(log(n))) && (19)\\
&= \Theta(nlog(n)) && (20)
\end{aligned}$$

2f)

In practice, a modified merge sort is chosen to execute in a faster time than a standard merge sort. Therefore, k should be chosen to to be the maximum size for which insertion sort is faster than merge sort.

3)

My implementation is in python; as I find it very close to pseudo-code in simple cases - with the added benefit that i can run it on sample cases. Please let me know if this is an issue and I can modify my approach in the future.

The algorithm loops over the nodes and adjacent nodes a maximum of only 1 time each; thus running in $\Theta(n+m)$. The out-degree is a simple count of the adjacency list, while the in-degree must be tracked and summed from all adjacency lists as the nodes are iterated.

```
def compute_degree(adj_list):
    """Computer the in-degree and out-degree of a given node"""
    out_counts = {}
    in_counts = {}

    for node in adj_list:
        out_counts[node] = len(adj_list[node])

        for other_node in adj_list[node]:
            if not other_node in in_counts:
                in_counts[other_node] = 0
            in_counts[other_node] += 1

    return out_counts, in_counts
```

4

For a path to be a valid binary search tree, each left subtree must be strictly smaller than it's root, and every right subtree must be strictly larger than it's root. For any of the paths given, if either of these conditions are broken then the path is not a valid binary search tree.

4.1)

This path is valid.

4.2)

This path is valid.

4.3)

This path is invalid. The node 912 appears on the left subtree of node 911, which violates the binary search conditions.

4.4)

This path is valid.

4.5)

This path is invalid. The node 299 appears on the right subtree of node 347, which violates the binary search conditions.

5)

Lets assume that the number of executions takes the form $2n + 1$. Solving the simple base case of $n = 0$:

$$2(0) + 1 = 1 \tag{21}$$

Which agrees with a tree of 0 nodes, for which only the initial conditional will be executed a single time.

For $n > 0$, execution will happen once on the root, pluse $2n + 1$ for the left subtree and $2n + 1$ for the right subtree:

$$\begin{aligned} N_{tot} &= 1 + (2n_l + 1) + (2n_r + 1) && (22)\\ &= 1 + (2n_l + 2n_r + 2) && (23)\\ &= 1 + 2(n_l + n_r + 1) && (24) \end{aligned}$$

Using the total number of nodes in the tree as $n = n_l + n_r + 1$:

$$N_{tot} = 1 + 2n \tag{25}$$