CGJ008 – inf102f18-mandatory0

Carl August Gjørsvik

Big-O Quiz

Function	f(n)	~	O()
A	2n+1	$\sim 2n$	$O(n)$
B	2n	$\sim 2n$	$O(n)$
C	$\frac{n(n-1)}{2}$	$\sim \frac{n^2}{2}$	$O(n^2)$
D	$\frac{n(n+1)}{2} + 2$	$\sim \frac{n^2}{2}$	$O(n^2)$
E	1	~ 1	$O(1)$
F	$\left\lceil \frac{n}{2} \right\rceil$	$\sim \frac{n}{2}$	$O(n)$
G	2n+2	$\sim 2n$	$O(n)$
H	$\lfloor \log_2 n \rfloor + 2$	$\sim \log_2 n$	$O(\log n)$
I	$\sum_{i=0}^{\lfloor \log_2 n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor$	$\sim 2n$	$O(n)$
J	$\sum_{i=0}^{\lfloor \log_2 n \rfloor} \left(2^i * \left\lfloor \frac{n}{2^i} \right\rfloor \right)$	$\sim n \log_2 n$	$O(n \log n)$
K	$\lceil \sqrt{n} \rceil$	$\sim \sqrt{n}$	$O(\sqrt{n})$
L	$\lfloor \log_2 n \rfloor + 1$	$\sim \log_2 n$	$O(\log n)$
M	$\sum_{i=0}^{n} \frac{n!}{i!}$ $alternatively$: $\lfloor e * n! \rfloor$	$\sim e * n!$	$O(n!)$
N	$\lceil \log_2 n \rceil$	$\sim \log_2 n$	$O(\log n)$
O	$\sum_{i=0}^{n} 3^i$	$\sim \frac{3^{n+1}}{2}$	$O(3^n)$
P	$n * \lfloor \log_2 n \rfloor + 2n$	$\sim n \log_2 n$	$O(n \log n)$
Q	$\lfloor \log_2 n \rfloor + 1$	$\sim \log_2 n$	$O(\log n)$
R	$n + 2\lceil \sqrt{n} \rceil + 1$	$\sim n$	$O(n)$
S	$\left\lceil \frac{n^3}{\lceil \sqrt{n} \rceil} \right\rceil * (\lceil \sqrt{n} \rceil + 1)$	$\sim n^3$	$O(n^3)$
T	$\left(\sum_{i=0}^{\lceil \sqrt{n} \rceil - 1} 2(n^2 - i) \right) + n^2 + \sum_{j=1}^{n} \lfloor \log_2 j \rfloor$	$\sim 2\sqrt{n}\, n^2$	$O(\sqrt{n}\, n^2)$
U	$1 + \sum_{i=0}^{\lfloor \log_2 n \rfloor} \left(\left\lceil \log \left\lfloor \frac{n}{2^i} \right\rfloor \right\rceil + 1 \right)$	$\sim \frac{\log n * \log 2n}{\log 4}$	$O((\log n)^2)$
V	$1 + \sum_{i=0}^{\lfloor \log_2 n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor$	$\sim 2n$	$O(n)$
W	$2n + \lfloor \log_2 n \rfloor + 1$	$\sim 2n$	$O(n)$
X	$2\left(\lfloor \log_2 n \rfloor + 2^{\lceil \log_2 n \rceil} + 1 \right)$	$\sim 2n$	$O(n)$

Y	$\left\lfloor \frac{\lfloor \sqrt{n} \rfloor}{2} \right\rfloor$	$\sim \frac{\sqrt{n}}{2}$	$O(\sqrt{n})$
Z	$2(n-1) + n\log(\log n)$	$\sim n\log(\log n)$	$O(n\log(\log n))$

Note: In the answer to functionM f(n), the first part is valid for n=0 and up. The alternative is valid for n=1 and up.

Union Find

c)

We would need to add a size[] array to keep track of the number of elements connected to a root. Using the size array to perform "weighted quick union". To maintain the oldest account associated with any given account, we can use a third array, lets call it oldest[]. Like the id[] array, the oldest[] array will first have every element pointing to itself. When performing union(p, q), we would need to find the roots of p and q, then set the one with the larger size as root of the other. Similarly, find the oldest account in the set containing p, and the set containing q, set the older of these as oldest associated of the younger. We would also need another find-method to find the oldest associated account of an element. This does not sound very effective, but possible.

```
public static void wightedQuickUnion(int p, int q) {
    int x = find(p);
    int y = find(q);

    if (size[x] > size[y]) id[y] = x;
    else id[x] = y;

    int Ox = findOldest(x);
    int Oy = findOldest(y);

    if (Ox < Oy) oldest[Oy] = Ox;
    else oldest[Ox] = Oy;
}

public static int find(int p) {
    while (id[p] != p) p = id[p];
    return p;
}

public static int findOldest(int p) {
    while (oldest[p] != p) p = oldest[p];
    return p;
}
```

If we were to solve the problem using node objects in a tree structure, one could perform weighted union, swapping the values of the root nodes if the lesser root is the top of the shorter tree.

d)

Scheme

Assuming we are allowed to make "stupid" requests, like union(p, q) where p = q.

Calling find(p) while p is root of itself, costs **1** array access

Calling find(p) while p is not root of itself costs **2h-1**, where h is the height of the tree (counting height as number of elements in the tree)

Calling union(p, q) where p is not a root, but q is root of itself (single element), costs 2h-1 for find(p) and 1 for find(q), and another for the actual union, resulting in **2h+1**

Calling union(p, q) where p = q and they are an element at the bottom of a worst-case tree (which we will create) costs find(p) = 2h-1, find(q) = 2h-1, and 1 for the union action. Result: 2(2h-1)+1 = **4h-1**

Now, the scheme would be to make a worst-case tree chaining elements, making the height equal to the number of elements. Example: For the 10 elements {0 to 9}, start with

Union(0,1)	1 <- 0
Union(1,2)	2 <- 1 <- 0
Union(2,3)	3 <- 2 <- 1 <- 0
…until union(8,9) ?	9 <- 8 <- … <- 1 <- 0

Now, at some point we would get a higher total number of array accesses if we stopped chaining elements, but rather called union(p, q) where p = q and it is the bottom of the tree, because this costs **4h-1**.

Example: after creating a worst-case tree of the elements 0 to 8

Union(8,9) = 2h+1 = **19**

Union(8,8) = 4h-1 = **35**

Analysis

Assuming $m \leq n$ (to begin with) we want to make $x < m$ calls to union(p, q), where p = 0, q=1 and both increase by 1 each step (creating a worst-case tree). Then $m - x$ calls to union(p, q) where p = q = x.

This can be expressed as:

$$\sum_{i=1}^{x} 2i + 1 + \sum_{i=x+1}^{m} 4x + 1$$

Using i as height for the first sum, because it increases with elements added, x as height in the second sum as we are no longer increasing the length of the tree.

The goal is to find x as a function of m.

The first sum can be written as $x(x + 2)$

The second sum can be written as – $(x - m)(4x - 1)$

Analysis of the following function $f(x) = x(x+2) - (x-m)(4x-1)$ shows that for any valid m, $f(x)$ is at its maximum when $x = \left\lfloor \frac{2}{3}m \right\rfloor$

Conclusion

Chaining elements in a worst-case tree until element $\left\lfloor \frac{2}{3}m \right\rfloor$ creates the optimal situation to make the maximum number of array accesses with the remaining actions, $\left\lfloor \frac{2}{3}m \right\rfloor \ to\ m$.

Now considering the case where $m > n$, the explanation above will be valid until $\left\lfloor \frac{2}{3}m \right\rfloor > n$. We can't add any more elements to the tree when we have added all n elements. Hence, for $\left\lfloor \frac{2}{3}m \right\rfloor > n$, we must add elements to the tree until n, and then for the remaining actions, call union(p, q) where p = q = the bottom of the tree.

$$f(n,m) = \sum_{i=1}^{min\left\{n,\left\lfloor \frac{2}{3}m \right\rfloor\right\}} (2i+1) + \sum_{min\left\{n,\left\lfloor \frac{2}{3}m \right\rfloor\right\}+1}^{m} \left(4 * min\left\{n, \left\lfloor \frac{2}{3}m \right\rfloor\right\} + 1\right)$$