Quick Sort Expected Number of Sub-Arrays of Lengths 0, 1 and 2

From https://stackoverflow.com/questions/30283079/what-is-the-expected-number-of-subarrays-of-size-0-1-and-2-when-quicksort-is-us

Re. Algorithms-4ed by Sedgewick and Wayne, Exercise 2.3.7 p303.

QuickSort will recursively partition the array into two smaller array at position k. k can be from 1 to n. Each k has the same probability of occurrence. Let `C0(n)` be the average number of appearances of 0-sized subsets, and `C1(n)`, `C2(n)` be the same for 1-sized and 2-sized subsets.

Apart from initial conditions, each satisfies:

```
C(n) = 1/n sum(C(k-1) + C(n-k) for k=1..n)
```

The two parts of the sum are the same but summed in the opposite order, so:

```
C(n) = 2/n sum(C(k-1) for k=1..n)
```

or

```
n*C(n) = 2*sum(C(k-1) for k=1..n)
```

Assuming neither `n` nor `n-1` are part of the initial conditions, we can simplify by subtracting `(n-1)C(n-1)` from both sides:

```
n*C(n) - (n-1)C(n-1) = 2*C(n-1)
```

or

```
C(n) = (n+1)/n * C(n-1)
```

Deriving results from the recurrence relation

We now have a recurrence relation `C(n)` which applies equally to `C0`, `C1` and `C2`.

For `C0`, we have initial conditions `C0(0)=1`, `C0(1)=0`. We compute `C0(2)` to get `1`, and then we can apply the simplified recurrence relation `C0(n) = (n+1)/n * C0(n-1)` for n>2 to get the general result `C0(n)=(n+1)/3`.

For `C1`, we have initial conditions `C1(0)=0`, `C1(1)=1`. As before, we compute `C1(2)` to get `1`, and apply the same procedure as for `C0` to get the general result `C1(n)=(n+1)/3`.

For `C2`, we have initial conditions `C2(0)=C2(1)=0`, and `C2(2)=1`. This time we compute `C2(3) = 1/3 * 2 * (C2(0) + C2(1) + C2(2)) = 2/3`. Then applying the simplified recurrence relation to infer the general result `C2(n)=(n+1)/4 * C2(3) = (n+1)/4 * 2/3 = (n+1)/6`.

Conclusion

We've shown the average number of appearances of 0-sized and 1-sized subarrays when quicksorting an array of size n is in both cases (n+1)/3. For 2-sized subarrays we've shown it's (n+1)/6.

This confirms your original observation that 2-sized subsets appear exactly half as often as 0 and 1-sized subsets, and gives an exact formula for the means.

edited May 18 '15 at 1:05 answered May 17 '15 at 4:35

shareimprove this answer

Paul Hankin

19.2k

invisal

7,312

Comments

That equation is termed "recurrence relation", and what exactly does it mean in your case? – Pavel

May 17 '15 at 4:42

@paulpaul1076, in QuickSort, you will recursively partition the array into two smaller array. k is position which QuickSort will split. But we don't kow where the k is, but we know that k can be from 1 to n and each case has a same probability. – invisal May 17 '15 at 4:45

I understand that we reduce the subarray size by 1 each time, so n becomes smaller..But I still don't understand the entire thing – Pavel May 17 '15 at 4:54 1

Looks correct to me, but how about subarrays of size 0? When C_0=1, you get (N+1) average occurrence, but that over-counts since each subarray of size 1 you encounter will count as producing 2 subarrays of size 0 (rather than none). But (N+1) - 2*(N+1)/2 = 0, which isn't right... –

Paul Hankin May 17 '15 at 5:02

Yeah, it is (N+1) for subarray of size 0. The reason that your count function does not count N+1 because that when you are C_1, you prevent C_0 from executing. In my equation, it does not prevent C_1 from going to C_0 – invisal May 17 '15 at 5:15

@invisal I think you should be able to correct the zero-subset size by subtracting twice the number of 1-subsets you expect. But that gives 0. What's wrong with my logic? – Paul Hankin May 17 '15 at 5:23

@Anonymous, it should be zero-subset subtracting the number of 1-subset that we expect. Because, based on equation, C_1 = C_0 + 0 (+ 0 because we don't count the C_1 occurrence). – invisal May 17 '15 at 5:29

@invisal I don't see why it's right to subtract once rather than twice, but I still like your answer ;) –

Paul Hankin May 17 '15 at 5:36

@Anonymous, yeah, sorry. My mind was gone wild because I was thinking too hard. I guess my math is still bad :(– invisal May 17 '15 at 5:44 1

I have figured out the mistake. For subarrays of size 2, you don't have nC(3)=(n+1)C(2) since C(2)

is fixed and not 2*(C(0)+C(1))/2. Instead C(3) = 2/3 by calculation, and you get C(n) = (n+1)/6 for

n>2. For subarrays of size 0 and size 1, C(2)=1 and C(n)=(n+1)/3 for n>2. – Paul Hankin May 17

'15 at 8:39

@Anonymous, Yeah, you are right. You are genius. You may edit my post. – invisal May 17 '15 at

12:19

I understand how you derived everything but can't understand how you get (N+1)/3 for C1 etc.

Also, usually in recurrence relations there should be a base case, which is C0, I guess, and I don't

understand what that is equal to, especially due to the fact that that's the thing that we need to

find. – Pavel May 17 '15 at 18:29

@invisal I heavily edited your proof, removing the images, fixed the bug, and tried to explain how C

relates to the number of 0, 1 and 2-sized subsets. I hope you don't mind the changes ;/ – Paul

Hankin May 18 '15 at 1:06

now i finally understand, since you write CY(X), makes more sense – Pavel May 18 '15 at 2:50

@Anonymous also you keep saying we compute C0(2) and C1(2) to get 1, if C0(1) = 0 how do we

get C0(2) = 1? from the formula we have that C0(2) = (2+1)/2 * C0(1) = 3/2 * 0 and that gives us 0.

Same for C1(2), C1(2) = 3/2 * 1 = 3/2, how did you get 1? – Pavel May 18 '15 at 3:14

@paulpaul1076 The simplified formula isn't guaranteed to hold when n or n-1 is one of the initial

conditions cases. The step where we subtract (n-1)C(n-1) is wrong if C(n-1) isn't 2*sum(C(k-1) for

k=1..n-1). – Paul Hankin May 18 '15 at 3:29

@Anonymous, let's look at C0(2) for example, say, we have an array { 1,2 }, if the first item is the

partitioning one, we get one 0 element array, if the second item is the partitioning one we also get

one 0 element array. so altogether we have 2/2 = 1, same for C1(2), but how did you get (n+1)/3?

for n > 2? – Pavel May 18 '15 at 3:39

@paulpaul1076 repeatedly apply the simplified formula until you get down to n=2. The fractions

(n+1)/n almost all cancel out. – Paul Hankin May 18 '15 at 3:55

@Anonymous, yeah, I've noticed that, thanks – Pavel May 18 '15 at 3:58