Data Structures

Randomized quick sort: Expected running time

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$
$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \textit{ is compared to } z_j\}$$

- We have to compute the quantity, $\Pr\{z_i \textit{ is compared to } z_j\}$
- Consider an input array A = {41, 11, 21, 51, 81, 61, 31, 71, 101, 91}
- Assume that the first call to Partition separates this array into two sets: {41, 11, 21, 51, 31} and {81, 71, 101, 91}
- An element from either of these sets will ever be compared with the elements in the other set?

Randomized quick sort: Expected running time

- Consider a set Z_{ij}
- If an element "x" s.t. $z_i < x < z_j$ is selected as the pivot, then can we compare z_i and z_j at any subsequent time?
- If z_i is chosen as the pivot, then z_i will be compared to all elements in Z_{ij} except for itself
- If z_j is chosen as the pivot, then z_j will be compared to all elements in Z_{ij} except for itself
- Observation 3:

Elements z_i and z_j are compared if the first element to be chosen as pivot from Z_{ij} is either z_i or z_j

Randomized quick sort: Expected running time

- Till the point we select an element from Z_{ij} as the pivot, all elements in Z_{ij} are in the same partition
- A = {41, 11, 21, 51, 81, 61, 31, 71, 101, 91}
- First two smallest elements: Z_{12} = {11, 21}
- A_1 = {41, 11, 21, 51, 31}; A_2 = {81, 71, 101, 91}
- A_{11} = {11, 21}; A_{12} = {41, 51}
- A_{111} = {11}
- Any elements from Z_{ij} is equally likely to be selected as the pivot
- There are j-i+1 elements in Z_{ij}
- The probability that any given element is the first one chosen as pivot is 1/(j-i+1)

Randomized quick sort: Expected running time

- Pr{z_i is compared to z_j} = Pr{z_i or z_j is the first pivot chosen from Z_{ij}}

  = Pr{z_i is the first pivot chosen from Z_{ij}} +

  Pr{z_j is the first pivot chosen from Z_{ij}}

  = 1/(j-i+1) + 1/(j-i+1)

  = 2/(j-i+1)

Randomized quick sort: Expected running time

- Using Pr{z_i is compared to z_j},

  E[X] = $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2/(j - i + 1)$

  = $\sum_{i=1}^{n-1} \sum_{k=1}^{n-i} 2/(k + 1)$

  < $\sum_{i=1}^{n-1} \sum_{k=1}^{n} 2/k$

  = $\sum_{i=1}^{n-1} \mathrm{O}(\log n)$

  E[X] is O(n log n)

- Using Randomized_Partition, the expected running time of quick sort algorithm is O(n log n) when the elements are distinct

Alternate analysis of expected running time

- Analyse the running time in terms of the expected running time of each individual recursive calls to randomised quicksort
- An array of n distinct elements
- The probability of selecting an element as pivot is: 1/n
- X_i = I{ith smallest element is selected as pivot}

$$= \begin{cases} 1 & \text{if ith smallest element is selected as pivot} \\ 0 & \text{otherwise} \end{cases}$$

- $E[X_i] = 1/n$

Alternate analysis of expected running time

- Suppose that ith smallest element is selected as the pivot
- The input array gets divided into two subarrays of size (i-1) and (n-i)
- The running time is:

$$T(n) = X_i(T(i-1) + T(n-i) + \theta(n))$$

- Expectation over all events and linearity of expectation

$$\begin{aligned} E[T(n)] &= E\left[\sum_{i=1}^{n} X_i(T(i-1) + T(n-i) + \theta(n))\right] \\ &= \sum_{i=1}^{n} E\left[X_i(T(i-1) + T(n-i) + \theta(n))\right] \\ &= \sum_{i=1}^{n} E[X_i](E[T(i-1)] + E[T(n-i)] + \theta(n)) \\ &= \sum_{i=1}^{n} (E[T(i-1)] + E[T(n-i)] + \theta(n))/n \end{aligned}$$

Alternate analysis of expected running time

$$E[T(n)] = \theta(n) + 1/n \sum_{i=1}^{n} (E[T(i-1)] + E[T(n-i)])$$

$$= \theta(n) + \frac{1}{n}\left(\sum_{i=1}^{n} E[T(i-1)] + \sum_{i=1}^{n} E[T(n-i)]\right)$$

$$= \theta(n) + \frac{1}{n}\left(\sum_{i=1}^{n} E[T(i-1)] + \sum_{i=1}^{n} E[T(i-1)]\right)$$

$$= \theta(n) + \frac{2}{n}\sum_{i=1}^{n} E[T(i-1)]$$

$$= \theta(n) + \frac{2}{n}\sum_{i=0}^{n-1} E[T(i)]$$

$$= \theta(n) + \frac{2}{n}\sum_{i=2}^{n-1} E[T(i)]$$

Alternate analysis of expected running time

$$E[T(n)] = \theta(n) + \frac{2}{n}\sum_{i=2}^{n-1} E[T(i)]$$

$$E[T(n)] = \frac{2}{n}\sum_{i=2}^{n-1} E[T(i)] + cn$$

$$nE[T(n)] = 2\sum_{i=2}^{n-1} E[T(i)] + cn^2$$

$$(n-1)\,E[T(n-1)] = 2\sum_{i=2}^{n-2} E[T(i)] + c(n-1)^2$$

Take difference:

$$nE[T(n)] - (n-1)\,E[T(n-1)] = 2E[T(n-1)] + 2cn - c$$

$$nE[T(n)] = (n+1)E[T(n-1)] + 2cn$$

Divide by n(n+1):

$$\frac{E[T(n)]}{n+1} = \frac{E[T(n-1)]}{n} + \frac{2c}{n+1}$$

Alternate analysis of expected running time

$$\frac{E[T(n)]}{n+1} = \frac{E[T(n-1)]}{n} + \frac{2c}{n+1}$$

$$\frac{E[T(n-1)]}{n} = \frac{E[T(n-2)]}{n-1} + \frac{2c}{n}$$

$$\frac{E[T(n-2)]}{n-1} = \frac{E[T(n-3)]}{n-2} + \frac{2c}{n-1}$$

$$\vdots$$

$$\frac{E[T(2)]}{3} = \frac{E[T(1)]}{2} + \frac{2c}{3}$$

Adding these equations:

$$\frac{E[T(n)]}{n+1} = \frac{E[T(1)]}{2} + 2c\sum_{i=3}^{n+1} \frac{1}{i}$$

Alternate analysis of expected running time

$\frac{E[T(n)]}{n+1} = \frac{E[T(1)]}{2} + 2c\sum_{i=1}^{n+1} \frac{1}{i} - \frac{3}{2}$

E[T(n)] is O(n log n)

Space complexity

- It is a function describing the amount of memory space an algorithms takes in terms of input size
- Can use natural units to measure the space requirement
- Number of integers used, number of fixed size structures
- The function is independent of bytes needed to represent a unit

Algorithm sum(x, y, z)

```
    r ← x + y + z
    return r
```

Space complexity

```
Algorithm sum1(A, n)
        r ← 0
        for i ← 0 to n-1 do
                r ← (r + A[i])
        return r
```

Space complexity

```
Algorithm matrixAdd(A, B, n)
    for i ← 0 to n-1 do
        for j ← 0 to n-1 do
            A[i,j] ← (A[i,j] + B[i,j])
```

Quick sort algorithm

Initial call: Quick_Sort(A, 0, n-1)

```
Quick_Sort(A, p, r)
    if(p < r)
        q = Partititon(A, p, r)
        Quick_Sort(A, p, q-1)
        Quick_Sort(A, q+1, r)
```

Another version of quicksort

- The second recursive call is not really necessary
- Can be avoided using an iterative control structure
- This technique is called tail recursion

```
Tail_Recursive_Quicksort(A, p, r)
        while p < r
                q ← Partition(A, p, r)
                Tail_Recursive_Quicksort(A, p, q-1)
                p ← q+1
```

Another version of quicksort

- Compilers use a stack to execute recursive calls
- This stack contains pertinent information including the parameter values for each recursive call
- Recent information is at the top
- When a procedure is called, its information is pushed onto stack
- When a procedure is terminated, its information is popped from stack
- Assume that the array parameters are represented by pointers
- The information for each procedure call: O(1)
- Stack depth: the maximum amount of stack space used at any time during a computation

Another version of quicksort

Initial call: Tail_Recursive_Quicksort(A, 0, n-1)

```
Tail_Recursive_Quicksort(A, p, r)
    while p < r
        q ← Partition(A, p, r)
        Tail_Recursive_Quicksort(A, p, q-1)
        p ← q+1
```

When does the stack depth is θ(n) on an n-element input array?

Another version of quicksort

- How can we modify the code so that worst-case stack depth is θ(log n)

```
Algorithm Tail_Recursive_Quicksort1(A, p, r)
        while p < r
                q ← Partition(A, p, r)
                if q < ⌊(p + r)/2⌋
                        Tail_Recursive_Quicksort1(A, p, q-1)
                        p ← q+1
                else
                        Tail_Recursive_Quicksort1(A, q+1, r)
                        r ← q-1
```