LECTURE 9 (September 24th)

Midterm 1 — Mon, September 30th, 7-9 pm

- No lecture on Thursday
  - Optional Midterm Review Session
  - Sample Exams on Jeff's website or previous years' offerings
- Exam will cover material upto HW3
  - Pre-requisites
  - Divide & Conquer, FFT
  - Dynamic Progmmaning
  - No Randomized Algorithms / Probability on Midterm 1
- 4 questions & 2 hours
  - Solutions will be short, should take 5-10 minutes to write
  - Clear and precise description of algorithm or pseudocode
    some justification, e.g., recurrence
    Run time
- One handwritten cheat sheet — one double-sided A4 page
- Conflict Exam on
  - Registration Form on Course Webpage
- DRES accommodation
  - Please send the DRES letter by
- No homework this week

Matching Nuts & Bolts

Introduced by Gregory Rawlins

We have a bag of n nuts and n bolts. All nuts are of different sizes and so are the bolts but for every nut there is exactly one matching bolt.

Task: Match every nut to its corresponding bolt

Only operation allowed: Compare a nut to a bolt and see if it fits or if the nut is bigger or the bolt is bigger

Cannot compare two nuts with each other or two bolts with each other

How would you do it?

<u>Brute Force</u> Compare every nut with every bolt – n^2 tries
One can save a factor of two since if we find a matching nut for a bolt we don't need to compare it to others, but it is still $\Theta(n^2)$ tries

This problem should remind you of sorting

If the bolts are sorted, we can pick a nut and find a matching bolt with binary search in $\log n$ time. Overall, this takes $O(n \log n)$ time to match all the nuts and bolts.

This process of assuming a solution to one problem and using it to solve another is called a reduction. So, we can reduce the nut-bolt matching problem to sorting.

It turns out that this task is equivalent to sorting. If all the nuts and bolts are matched, then we can also sort in $O(n \log n)$ time because if we want to compare two nuts with each other, we can compare them to their corresponding bolt. So, we can run mergesort to sort in $O(n \cdot \log n)$ time.

So, to solve the task, let's try a sorting algorithm e.g. Mergesort. Can we do mergesort with nuts & bolts?

Recall that mergesort does the following to an input array:

1. Split it into two halves & sort each half
2. Merge the two sorted halves

<u>Why can't we do this here</u>?

Problem comes from step 1 of mergesort.

We can split the nuts into two parts but to recurse, we need to split the bolts into two parts that match the partition of the nuts. Not obvious how to do that!

So, this doesn't work, as there is no way to recurse.

<u>What about Quicksort</u>?

At a minimum, we can make quicksort work

- pick a bolt, call it the pivot bolt
- use that bolt to partition the nuts into three parts – those that are bigger than the pivot bolt, those that are smaller and the matching nut
- then we can partition the bolts, using the nut that matches the pivot bolt

Now, we can recurse since we know in one part all of the nuts/bolts are smaller than the pivot, and in the other, larger than the pivot

If we try to do this deterministically, this is $\Theta(n^2)$ in the worst-case, just like basic quicksort since everytime in the recursion, we might get unlucky and the problem splits into two unbalanced parts

For quicksort, we can use median of medians to pick a clever pivot and reduce the running time to $O(n \cdot \log n)$ but there seems to be no way of doing that here

It is possible to do it deterministically in $O(n \cdot \log n)$ time, but the algorithm is quite complicated and it took a lot of work to find it.

Picking a random pivot, also gives an $O(n \cdot \log n)$ time randomized algorithm that is not so difficult to analyze.

Randomized Algorithm to Match Nuts & Bolts

1. Pick a random pivot
2. Partition nuts/bolts into bigger & smaller parts & matched pair
3. Recurse on the two sub-problems

How many tests do we need to perform for 2 ?

$2n-1$ tests, compare the pivot bolt to all n nuts
compare the matched nut to everything except the pivot bolt

Let $T(n)$ = running time for n nuts & bolts

Then, $T(n) = 2n-1 + T(k-1) + T(n-k)$ for some value of k

If we were looking at the worst-case behavior, then we would take a max over all k. But since we chosen the pivot randomly, we are interested in the expected value of the running time, since k is random

So, $\mathbb{E}[T(n)] = 2n-1 + \mathbb{E}_K\left[\mathbb{E}[T(K-1)] + \mathbb{E}[T(n-K)]\right]$

Since the pivot bolt is equally likely to be the smallest, second smallest or k^{th} smallest for any k,

$\mathbb{P}[K=k] = \frac{1}{n} \quad \forall k$, so our recurrence becomes

$$\mathbb{E}[T(n)] = 2n-1 + \frac{1}{n}\sum_{k=1}^{n}\left(\mathbb{E}[T(k-1)] + \mathbb{E}[T(n-k)]\right)$$

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

How to solve this recurrence? Such recurrences are called, full history recurrences and there's a trick to solving them which you can read in the lecture notes, but here we will try something else

Heuristic Calculation — Not a proof!

What are the bad cases for quicksort? When sub-problems are very unbalanced

Call a pivot good if $\frac{n}{4} < k < \frac{3n}{4}$. Then, $\mathbb{P}[\text{pivot is good}] = \frac{1}{2}$

So, $\mathbb{E}[T(n)] = \frac{1}{2}\,\mathbb{E}[T(n) \mid \text{good}] + \frac{1}{2}\,\mathbb{E}[T(n) \mid \text{bad}]$

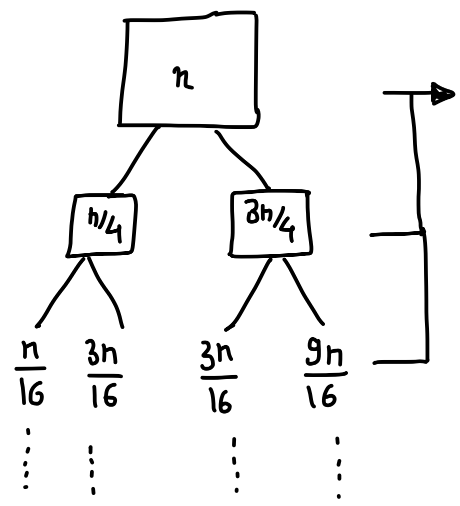
Now, we can make the unjustified assumption that the running time only increases the more unbalanced our pivot. So, even when the pivot is good, the worst-case occurs when $k = \frac{n}{4}$ or $\frac{3n}{4}$. Similarly, when pivot is bad, then the worst-case occurs when one sub-problem has size zero.

With this "heuristic" assumption,

$$\mathbb{E}[T(n)] \leq 2n - 1 + \frac{1}{2}\left[\mathbb{E}\left[T\left(\frac{n}{4}\right)\right] + \mathbb{E}\left[T\left(\frac{3n}{4}\right)\right]\right] + \frac{1}{2}\underbrace{\mathbb{E}[T(n-1)]}_{\leq\, \mathbb{E}[T(n)]}$$

$$\Rightarrow \mathbb{E}[T(n)] \leq 4n - 2 + \mathbb{E}\left[T\left(\frac{n}{4}\right)\right] + \mathbb{E}\left[T\left(\frac{3n}{4}\right)\right]$$

We can solve this by using recursion trees



Total work at each level $= n$

$\#$ levels $= \log_{4/3} n$ $= O(n \cdot \log n)$

Thus, heuristically $\mathbb{E}[T(n)] = O(n \cdot \log n)$ → This relies on an unjustified assumption which would be true if $\mathbb{E}[T(n)]$ is a convex function of n. This turns out to be true but proving it is not easy.

So, to prove it rigorously, we introduce a different method which illustrates one of the most useful ideas in the analysis of randomized algorithms — decomposing the running time as a sum of zero-one random variables, called indicator variables.

Rigorous Analysis

Let $X_{ij} = \begin{cases} 1 & \text{if } i^{th} \text{ smallest bolt is compared to } j^{th} \text{ smallest nut} \\ 0 & \text{o/w} \end{cases}$

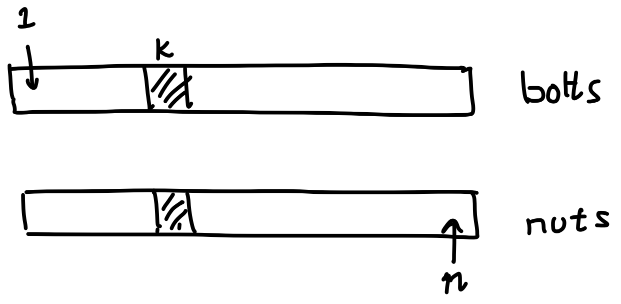
Then, $T(n) = \sum_{ij} X_{ij}$

$$\mathbb{E}[T(n)] = \sum_{ij} \mathbb{E}[X_{ij}]$$
$$= \sum_{ij} \left(1 \cdot \mathbb{P}[X_{ij} = 1] + 0 \cdot \mathbb{P}[X_{ij} = 0]\right)$$
$$= \sum_{ij} \mathbb{P}[X_{ij} = 1]$$

Let's look at a few cases to see what $\mathbb{P}[X_{ij} = 1]$ is.

- $\mathbb{P}[X_{7,7} = 1] = 1$ since i^{th} bolt must be compared to i^{th} nut in the end
- $\mathbb{P}[X_{1,n} = 1] = \frac{2}{n}$ $\Leftarrow$ There are several cases here:

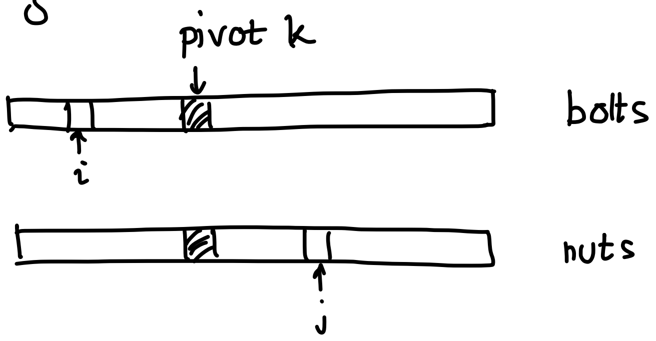
(1) pivot is the 1^{st} bolt $\Rightarrow$ $X_{1n} = 1$ $\Rightarrow$ probability $\frac{1}{n}$
(2) pivot is the n^{th} bolt $\Rightarrow$ $X_{1n} = 1$ $\Rightarrow$ probability $\frac{1}{n}$
(3) pivot is k^{th} bolt where $1 < k < n$ $\Rightarrow$ $X_{1n} = 0$



If k is selected, the left and right problems are solved separately and no bolt on the top left is compared to a nut in the bottom right

Intuitively, if a bolt and nut is closer in rank, it has a higher chance of being compared. For i^{th} bolt and j^{th} nut, if $|i-j|$ is small, then probability of being compared should be large.

- $\mathbb{P}[X_{ij} = 1] = \boxed{?}$

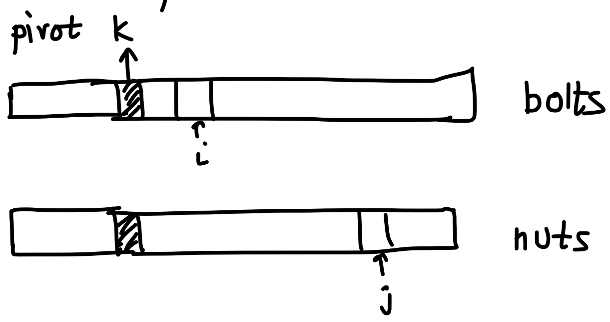


Again there are several cases here,

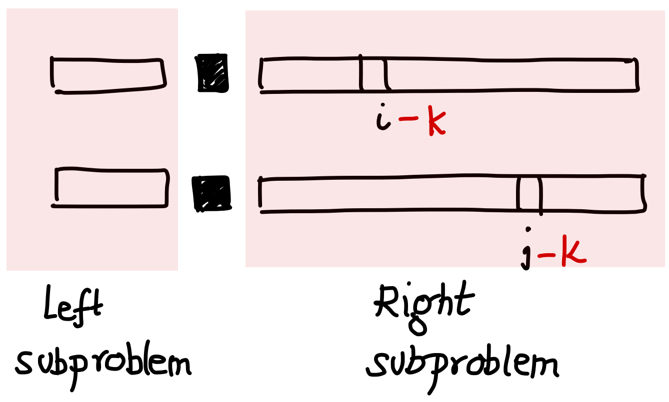
Case $k < i$	??
Case $k = i$	$X_{ij} = 1$
Case $i < k < j$	$X_{ij} = 0$
Case $k = j$	$X_{ij} = 1$
Case $k > j$	??

$\Rightarrow$ What happens in the top and bottom cases?

Consider the top case (since the bottom case is symmetric)



This will be divided into two subproblems in recursion – the left & right of the pivot



For the right subproblem, the original nut bolt pair we were interested in are now the $i-k$ & $j-k$ smallest respectively

Note that $|i-j| = |(i-k)-(j-k)|$ so the difference in their ranks has not changed

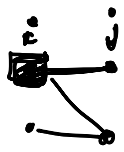
So, in this case where $k<i$, we recurse (or $k>j$)

Case $k<i$	Recurse
Case $k=i$	$X_{ij}=1$
Case $i<k<j$	$X_{ij}=0$
Case $k=j$	$X_{ij}=1$
Case $k>j$	Recurse

If the first pivot is chosen from $\{i, \ldots, j\}$, then there are $|j-i|+1$ cases where we either compare i & j or split i and j & never compare. Out of these there are exactly two cases where $X_{ij}=1$

So, $\mathbb{P}[X_{ij}=1 \mid k=i \text{ or } i<k<j \text{ or } k=j] = \frac{2}{|j-i|+1}$

To handle the recursive cases, we can use induction



Hypothesis For all $n \geq 2$, $\mathbb{P}[X_{ij}] = \frac{2}{|j-i|+1}$ for $j \neq i$

Base Case $n=2$ $\mathbb{P}[X_{ij}] = 1$ for $j \neq i$

Inductive Case

$$\mathbb{P}[X_{ij}=1] = \underbrace{\mathbb{P}\left[X_{ij}=1 \,\middle|\, \begin{matrix} k=i \text{ or} \\ i<k<j \text{ or} \\ k=j \end{matrix}\right]}_{=\frac{2}{|j-i|+1}} \cdot \underbrace{\mathbb{P}\left[\begin{matrix} k=i \text{ or} \\ i<k<j \text{ or} \\ k=j \end{matrix}\right]}_{:=p}$$

$$+ \underbrace{\mathbb{P}[X_{ij}=1 \mid k<i]}_{\text{By induction}} \cdot \underbrace{\mathbb{P}[k<i]}_{:=q} + \underbrace{\mathbb{P}[X_{ij}=1 \mid k>j]}_{\text{By induction}} \cdot \underbrace{\mathbb{P}[k>j]}_{:=r}$$

$$= \frac{2}{|(j-k)-(i-k)|+1} = \frac{2}{|j-i|+1}$$

Overall, $\mathbb{P}[X_{ij}=1] = \frac{2}{|j-i|+1} \underbrace{(p+q+r)}_{=1} = \frac{2}{|j-i|+1}$

Therefore,

$$\mathbb{E}[T(n)] = \sum_{ij} \mathbb{P}[X_{ij}=1]$$

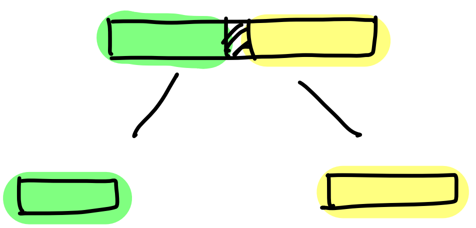
$$= n + \sum_{i \neq j} \mathbb{P}[X_{ij}=1]$$

$$= n + 2 \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{|j-i|+1}$$

$$= 4n H_n - 7n + 4H_n$$ where $H_n = n^{th}$ Harmonic number $\approx \log n$

Another piece of intuition to help you think about this algorithm is the following

Normally, when we think of our recursive algorithm, we think of it as follows



Pick a pivot

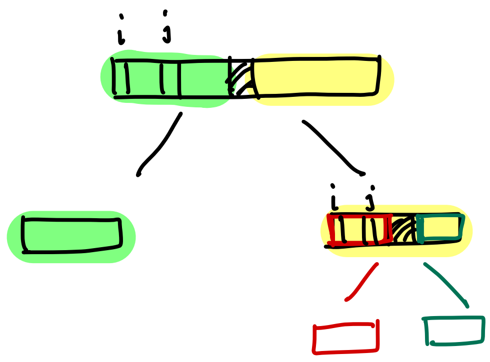
Recursively solve the left subproblem first
This means picking a random pivot in the green part and recursing

After we have solved the left subproblem, solve the right one

But you can also consider a slightly different version of the algorithm which does the same steps in a different order

The new algorithm has n phases: in each phase, it picks a random pivot bolt from all the bolts that were not chosen as a pivot before

The first phase works the same as above, but in the second phase the algorithm choses a uniformly random bolt in either the green or yellow subproblem and splits that part further. For example, the following could happen:



Split whatever region the randomly chosen bolts happen to land in!

Now, we can follow i^{th} nut and j^{th} bolt down the tree and this makes it a bit more intuitive that it only depends on $|j-i|$