Last time

- What are randomized algorithms
- Probabilities
- Contention resolution algorithms
- Finding the global minimum cut
Today

- Linearity of expectations
- MAX 3-SAT
- Finding the median and Sorting
13.3 Linearity of Expectation
Quality and quantity

So far, qualitative analysis

• Identifying “bad” events and bounding their probabilities

Quantitative

• Parameters associated with algorithm behavior
  • Running time, solution quality

• Determine/analyze the expected size of these parameters
  • Given random choices of the algorithm

• Random variable $X: \Omega \rightarrow \text{Nat}$, $X^{-1}(j)$ - event
  • $X^{-1}(j)$ all sample points taking value $j$
  • $\Pr[X=j]$ ($=\Pr[X^{-1}(j)]$) $\rightarrow X$ random variable
Expectation

Given a discrete random variable $X$, its expectation $E[X]$ (average value assumed by $X$) is defined by:

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j]$$

Waiting for a first success. Coin is heads with probability $p$ and tails with probability $1-p$. How many independent flips $X$ until first heads?

$$E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{\infty} j \cdot (1-p)^{j-1} p = \frac{1}{p}$$
Expectation: Two Properties

Useful property. If $X$ is a 0/1 random variable, $E[X] = \Pr[X = 1]$.

Pf.

\[ E[X] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j] = \sum_{j=0}^{1} j \cdot \Pr[X = j] = \Pr[X = 1] \]

not necessarily independent

Linearity of expectation. Given two random variables $X$ and $Y$ defined over the same probability space, $E[X + Y] = E[X] + E[Y]$.

- Decouples a complex calculation into simpler pieces.
Guessing Cards

Game. Shuffle a deck of n cards; turn them over one at a time; try to guess each card.

Memoryless guessing. No psychic abilities; can't even remember what's been turned over already. Guess a card from full deck uniformly at random.

Claim. The expected number of correct guesses is 1.

Pf. (surprisingly effortlessly using linearity of expectation)

• Let $X_i = 1$ if $i^{th}$ prediction is correct and 0 otherwise.
• Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.  
• $E[X_i] = \Pr[X_i = 1] = 1/n$.  
• $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/n = 1$.  

\[ \text{linearity of expectation} \]
Guessing Cards

Game. Shuffle a deck of n cards; turn them over one at a time; try to guess each card.

Guessing with memory. Guess a card uniformly at random from cards not yet seen.

Claim. The expected number of correct guesses is $\Theta(\log n)$.

Pf.
- Let $X_i = 1$ if $i^{th}$ prediction is correct and 0 otherwise.
- Let $X = \text{number of correct guesses} = X_1 + \ldots + X_n$.
- $E[X_i] = \Pr[X_i = 1] = 1 / (n - i + 1)$.
- $E[X] = E[X_1] + \ldots + E[X_n] = 1/n + \ldots + 1/2 + 1/1 = H(n)$. ▪

\[ \ln(n+1) < H(n) < 1 + \ln n \]

linearity of expectation
Coupon Collector

**Coupon collector.** Each box of cereal contains a coupon. There are \( n \) different types of coupons. Assuming all boxes are equally likely to contain each coupon, how many boxes before you have \( \geq 1 \) coupon of each type?

**Claim.** The expected number of steps is \( \Theta(n \log n) \).

**Pf.**

- Phase \( j \) = time between \( j \) and \( j+1 \) distinct coupons.
- Let \( X_j \) = number of steps you spend in phase \( j \).
- Let \( X = \) number of steps in total = \( X_0 + X_1 + \ldots + X_{n-1} \).

\[
E[X] = \sum_{j=0}^{n-1} E[X_j] = \sum_{j=0}^{n-1} \frac{n}{n-j} = n \sum_{i=1}^{n} \frac{1}{i} = n H(n)
\]

\[
\text{prob of success} = \frac{n-j}{n} \quad \Rightarrow \text{expected waiting time} = \frac{n}{n-j}
\]
Conditional expectation

Expectation of a random variable conditioned by a certain event

- $X$ random variable
- $F$ event s.t. $\Pr[F] > 0$
- $E[X|F]$ - conditional expectation of $X$, given $F$ (expected value of $X$ only over the part of the sample space corresponding to $F$)

$$E[X \mid F] = \sum_{j=0}^{\infty} j \cdot \Pr[X = j \mid F]$$
13.4 MAX 3-SAT
Maximum 3-Satisfiability

**MAX-3SAT.** Given 3-SAT formula, find a truth assignment that satisfies as many clauses as possible.

\[
\begin{align*}
C_1 &= x_2 \lor \overline{x_3} \lor \overline{x_4} \\
C_2 &= x_2 \lor x_3 \lor x_4 \\
C_3 &= \overline{x_1} \lor x_2 \lor x_4 \\
C_4 &= \overline{x_1} \lor x_2 \lor \overline{x_3} \\
C_5 &= x_1 \lor x_2 \lor x_4
\end{align*}
\]

**Remark.** NP-hard search problem.

**Simple idea.** Flip a coin, and set each variable true with probability \(\frac{1}{2}\), independently for each variable.
Claim. Given a 3-SAT formula with $k$ clauses, the expected number of clauses satisfied by a random assignment is $7k/8$.

Pf. Consider random variable $Z_j = \begin{cases} 1 & \text{if clause } C_j \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases}$

- Let $Z = \text{number of satisfied clauses}$

\[
E[Z_i] = \Pr[Z_i = 1] = 1 - \frac{1}{8} = \frac{7}{8}
\]

\[
\Pr[Z_i = 0] = \left( \frac{1}{2} \right)^3
\]

\[
E[Z] = \sum_{j=1}^{k} E[Z_j] = \sum_{j=1}^{k} \frac{7}{8} = \frac{7}{8} k
\]
The Probabilistic Method

**Corollary.** For any instance of $3$-SAT, there exists a truth assignment that satisfies at least a $7/8$ fraction of all clauses.

**Pf.** Random variable is at least its expectation at some point in time. ▪

**Probabilistic method.** We showed the existence of a non-obvious property of $3$-SAT by showing that a random construction produces it with positive probability!
Claim. Every instance of 3-SAT with at most 7 clauses is satisfiable.

Pf. $k \leq 7 \iff k < 8 \iff 8k - 7k < 8 \iff 8k - 8 < 7k \iff k - 1 < 7k/8 \iff 7 \geq k \geq k*7/8 > k - 1$

The number of satisfied clauses is at least $k*7/8$ and needs to be an integer number, $k*7/8 > k - 1$, hence the number of satisfied clauses is $k (7)$
Maximum 3-Satisfiability: Analysis

Q. Can we turn this idea into a 7/8-approximation algorithm? In general, a random variable can almost always be below its mean.

Lemma. The probability that a random assignment satisfies $\geq 7k/8$ clauses is at least $1/(8k)$.

Pf. Let $p_j$ be probability that exactly $j$ clauses are satisfied; let $p$ be probability that $\geq 7k/8$ clauses are satisfied.

$$\frac{7}{8}k = E[Z] = \sum_{j \geq 0} j p_j$$

$$= \sum_{j < 7k/8} j p_j + \sum_{j \geq 7k/8} j p_j$$

$$\leq \left( \frac{7k}{8} - \frac{1}{8} \right) \sum_{j < 7k/8} p_j + k \sum_{j \geq 7k/8} p_j$$

$$\leq \left( \frac{7}{8} k - \frac{1}{8} \right) \cdot 1 + k p$$

Rearranging terms yields $p \geq 1 / (8k)$. ▪
13.5 Randomized Divide-and-Conquer
Divide-and-Conquer

Divide-and-conquer.
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.
- Break up problem of size n into two equal parts of size $\frac{1}{2}n$.
- Solve two parts recursively.
- Combine two solutions into overall solution in linear time.

Consequence.
- Brute force: $n^2$.
- Divide-and-conquer: $n \log n$.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar
Randomized Divide-and-Conquer

- We perform the divide step using randomization
- We use expectations of random variables to analyze the time spent on recursive calls
- Problems
  - Finding the median of n numbers
  - Quicksort
The problem of **median finding**

$S = \{a_1, \ldots, a_n\}$; median $m$ is the number in the middle position if we sorted $S$

**Example.** (assume all numbers distinct)

$S = \{5, 4, 2, 16, 24, 2.5, 91, 7, 20\}$, $m = 7$

$S_{\text{sorted}} = \{2, 2.5, 4, 5, 7, 16, 20, 24, 91\}$

**Technicality.** Median $m$ is the $k^{th}$ largest element

- $n$ odd: $k = (n+1)/2$
- $n$ even: $k = n/2$
How long does it take to find the median?

- If S sorted, $O(n \times \log n)$
- Is it necessary to sort S??

- We present algorithm that does it in expected $O(n)$!!
How long does it take to find the median?

- If $S$ sorted, $O(n \log n)$
- Is it necessary to sort $S$?

- We present algorithm that does it in expected $O(n)$!!
Selecting splitters

• Median finding $\rightarrow$ Selection
• \textbf{Select}(S,k): returns the $k^{th}$ largest element in $S$
  • \textbf{Select}(S,n/2) or \textbf{Select}(S, n+1/2): median
  • \textbf{Select}(S,1): min
  • \textbf{Select}(S,n): max
• We want to find algorithm for \textbf{Select}(S,k) in expected time $O(n)$
  • Choose $a_i$ splitter
  • Form $S^+ = \{a_j | a_j > a_i\}$ and $S^- = \{a_j | a_j < a_i\}$
  • Determine which of $S^+$ and $S^-$ contains $k^{th}$ largest
  • Iterate on that one
Algorithm for \textbf{Select}(S,k):

Choose a splitter $a_i \in S$

For each element $a_j \in S$, $j \neq i$

- Put $a_j$ in $S^+$ if $a_j > a_i$
- Put $a_j$ in $S^-$ if $a_j < a_i$

endfor

If $|S^-| = k - 1$ then

- $a_i$ the desired answer

Else if $|S^-| \geq k$ then

- the $k^{th}$ largest element is in $S^-$
  - recursively call Select($S^-$, $k$)

Else assume $|S^-| = l < k - 1$

- the $k^{th}$ largest element is in $S^+$
  - recursively call Select($S^+$, $k - 1 - l$)

Endif

\textbf{Algorithm terminates and produces the right result}
“Good” Splitter

\[ T(n) \] - worst case running time; how does it depend on the splitter choice?

- Splitter selection: linear time
- \[ T(n) = \text{linear time} + \text{time for recursive call} \]
- Important: splitter reduces size of set to be considered

\[ \Rightarrow S^- \text{ and } S^+ \text{ should produce sets of } \approx \text{equal sizes} \]
Best splitter

The median itself.

- \( T(n) \leq T(n/2) + cn \Rightarrow T(n) = O(n) \)
- \( T(n) = cn + cn/2 + cn/4 + \ldots \)

\[
 cn \cdot \sum_{r=0}^{\log_2 n - 1} \left(\frac{1}{2}\right)^r = cn \cdot \frac{1 - \frac{1}{2^{\log_2 n}}}{1 - \frac{1}{2}} = 2cn \cdot \left(1 - \frac{1}{n}\right) = 2cn - 2c
\]
Well-centered splitter

$\varepsilon > 0$, splitter $a_i$ produces sets with at least $\varepsilon n$ elements both larger and smaller than $a_i$

$\Rightarrow$ size of sets in recursive calls decreases by at least $(1 - \varepsilon)$ each time

$\Rightarrow T(n) \leq T((1 - \varepsilon)n) + cn \Rightarrow T(n) = O(n)$

$$cn \cdot \sum_{r=0}^{\log_2 n} (1 - \varepsilon)^r \leq cn \cdot \sum_{r=0}^{\infty} (1 - \varepsilon)^r = cn \cdot \frac{1}{\varepsilon}$$
**Off-center splitter**

**Example.** We always choose the min as splitter.

- $T(n) \leq T(n-1) + cn \Rightarrow T(n) = O(n^2)$ !

\[
T(n) \leq cn + c(n - 1) + c(n - 2) + ... = \frac{cn(n + 1)}{2}
\]
Random splitters

Any well-centered splitter will do. We select the splitter by:

Choose a splitter $a_i \in S$ uniformly at random

Intuition: many elements are reasonably well-centered $\Rightarrow$ likely to end up with a good splitter when selecting at random
Analysis

**Idea**: we expect the size of the set under consideration to go down by a fixed constant fraction every iteration, in order to get a convergent series (and a linear bound)

Phase $j$: algorithm is in phase $j$ while the size $s$ of set under consideration is

$$n \cdot \left(\frac{3}{4}\right)^{j+1} \leq s \leq n \cdot \left(\frac{3}{4}\right)^j$$
Time spent by algorithm in phase \( j \)

Element \( e \) is **central** in set under consideration if:

\[
\{a_j | a_j < e\} \quad \text{and} \quad \{a_k | a_k > e\} \quad \text{are so that}
\]

\[
| \{a_j | a_j < e\} | \geq n/4, \quad | \{a_k | a_k > e\} | \geq n/4
\]

When chosen splitter is a central element

⇒ at least a quarter of the set under consideration is thrown away

⇒ set shrinks by at least a factor of \( \frac{3}{4} \)

⇒ current phase ends
How many iterations in phase j?

Element $e$ is central in set under consideration if:

$$\{a_j | a_j < e\} \text{ and } \{a_k | a_k > e\}$$

are so that

$$|\{a_j | a_j < e\}| \geq n/4, \ |\{a_k | a_k > e\}| \geq n/4$$

⇒ half of the elements are central in set under consideration

⇒ $\Pr[\text{choosing central element as splitter}] = 1/2$

⇒ expected nr of iterations before central element is found is $2!$ (slide 6)
Expected running time of Select(S, k) is $O(n)$

$X$ - random variable, denoting number of steps taken by algorithm

$X = X_0 + X_1 + X_2 + \ldots$, with $X_j \rightarrow$ expected number of steps spent in phase $j$

During phase $j$: set has size $\leq n\left(\frac{3}{4}\right)^j \Rightarrow$ at most $cn\left(\frac{3}{4}\right)^j$ steps in one iteration, at most 2 iterations

$\Rightarrow E[X_j] \leq 2cn\left(\frac{3}{4}\right)^j$

Hence

$$E[X] = \sum_j E[X_j] \leq \sum_j 2cn\left(\frac{3}{4}\right)^j \leq 2cn \cdot \frac{1}{1 - \frac{3}{4}} = 8cn$$
Randomized quicksort

- $S = \{a_1, \ldots, a_n\}$; we need to sort it
- Choose splitter from input set $S$
- Separate $S$ into two sets $S^+$ and $S^-$
- Difference: we sort recursively both sets $S^+$ and $S^-$ and glue the sorted sets together with splitter in between
Quicksort(\(S\)):

\(\text{If } |S| \leq 3 \text{ then} \)

Sort \(S\)

Output the sorted list

\(\text{Else} \)

Choose a splitter \(a_i \in S\) uniformly at random

For each element \(a_j \in S\), \(j \neq i\)

Put \(a_j\) in \(S^+\) if \(a_j > a_i\)

Put \(a_j\) in \(S^-\) if \(a_j < a_i\)

endfor

Recursively call Quicksort(\(S^-\)) and Quicksort(\(S^-\))

Output the sorted \(S^-\), then \(a_i\), then the sorted \(S^+\)

Endif
Worst-case running time

**Example.** We always choose the min as splitter.

- \( T(n) \leq T(n-1) + cn \Rightarrow T(n) = O(n^2) \)

\[
T(n) \leq cn + c(n-1) + c(n-2) + ... = \frac{cn(n+1)}{2}
\]

- This is worst-case running time for Quicksort.
Splitters

The median itself: best.

- \( T(n) \leq 2T(n/2) + cn \Rightarrow T(n) = O(n \log n) \)
- \( T(n) = cn + 2cn/2 + 4cn/4 + \ldots = \log_2 n \cdot cn \)

**Expected running time**

- We will show that it is bounded by \( O(n \log n) \)
ModifiedQuicksort(S):
If $|S| \leq 3$ then
    Sort S
    Output the sorted list
Else
    While no central splitter has been found
        Choose a splitter $a_i \in S$ uniformly at random
        For each element $a_j \in S$, $j \neq i$
            Put $a_j$ in $S^+$ if $a_j > a_i$
            Put $a_j$ in $S^-$ if $a_j < a_i$
        EndFor
        If $|S^-| \geq \frac{|S|}{4}$ and $|S^+| \geq \frac{|S|}{4}$ then
            $a_i$ is a central splitter
        EndIf
    EndWhile
    Recursively call Quicksort($S^-$) and Quicksort($S^-$)
    Output the sorted $S^-$, then $a_i$, then the sorted $S^+$
Endif
**Modified Quicksort Analysis**

The expected running time for the algorithm on a set $S$, excluding the time spent on recursive calls is $O(|S|)$.

Sub-problem of type $j$ if the size $s$ of the set under consideration is

$$n \cdot \left(\frac{3}{4}\right)^{j+1} \leq s \leq n \cdot \left(\frac{3}{4}\right)^j$$

$\Rightarrow$ the expected time spent on a sub-problem of type $j$, excluding recursive calls, is $O(n^*\left(\frac{3}{4}\right)^j)$. 
How many sub-problems of type $j$?

Sub-problem of type $j$ if the size $s$ of the set under consideration is

$$n \cdot \left(\frac{3}{4}\right)^{j+1} \leq s \leq n \cdot \left(\frac{3}{4}\right)^j$$

Denote number of sub-problems by $x \Rightarrow$

$$n \geq x \cdot s \geq x \cdot n \cdot \left(\frac{3}{4}\right)^j \Rightarrow$$

$$x \leq (4/3)^j$$
Expected running time for ModifiedQuicksort

At most \((4/3)^j\) sub-problems of type \(j\)

The expected time spent on each is \(O(n^{(\frac{3}{4})^j})\).

Hence, the expected time spent on sub-problems of type \(j\) is \(O(n)\).

Number of different types \(\leq \log_{4/3}n = O(\log n)\)

Hence, the expected running time is \(O(n^{\log n})\)