$$x \cdot y = \begin{cases} 0 & \text{if } x = 0 \\ \lfloor x/2 \rfloor \cdot (y + y) & \text{if } x \text{ is even} \\ \lfloor x/2 \rfloor \cdot (y + y) + y & \text{if } x \text{ is odd} \end{cases}$$

```
PEASANTMULTIPLY(x, y):
  if x = 0
      return 0
  else
      x' ← ⌊x/2⌋
      y' ← y + y
      prod ← PEASANTMULTIPLY(x', y')    ⟨⟨Recurse!⟩⟩
      if x is odd
          prod ← prod + y
      return prod
```

Bubble Sort Proof:

Inversion: Two indices i & j s.t.

$A[i] > A[j]$ & $i < j$

correct for k'th inversions

$k :=$ # inversions in A, IH: alg is

If no inversions, alg does nothing

but array is already sorted ✓

O.W. alg swaps $A[i]$ & $A[i+1]$,

reducing # inversions by one

so remaining $k-1$ swaps do sort

array

QUICKSORT($A[1..n]$):
  if ($n > 1$)
    Choose a pivot element $A[p]$
    $r \leftarrow$ PARTITION(A, p)
    QUICKSORT($A[1..r-1]$)   ⟨⟨*Recurse!*⟩⟩
    QUICKSORT($A[r+1..n]$)   ⟨⟨*Recurse!*⟩⟩

PARTITION($A[1..n], p$):
  swap $A[p] \leftrightarrow A[n]$
  $\ell \leftarrow 0$ ⟨⟨*#items < pivot*⟩⟩
  for $i \leftarrow 1$ to $n-1$
    if $A[i] < A[n]$
      $\ell \leftarrow \ell + 1$
      swap $A[\ell] \leftrightarrow A[i]$
  swap $A[n] \leftrightarrow A[\ell+1]$
  return $\ell + 1$

Claim: After ith iteration,

all of $A[1..\ell] < A[n]$

all of $A[\ell+1..i] \geq A[n]$,

Assume after iteration $i' < i$

with associated ℓ',

$A[1..\ell'] < A[n]$

$A[\ell'+1..i'] \geq A[n]$

Suppose $i \geq 1$.

Start iteration i.

If $A[i] \geq A[n]$,

uses IH: $A[1 \ldots \ell] < A[n]$ still

uses IH: $A[\ell+1 \ldots i-1] \geq A[n]$ still

& we just confirmed

$A[i] \geq A[n]$

If $A[i] < A[n]$...

$A[1 \ldots \text{old } \ell] < A[n]$

new $A[\ell]$ is old $A[i] < A[n]$

If $i > \ell$ before

we placed old $A[\ell+1] \geq A[n]$

into new $A[i]$

& $A[\text{new } \ell+1 \ldots \text{new } i-1] \geq A[n]$

still

if $i = l$ before, then $A[\text{new } l+1 \ldots \text{new } i]$ is empty so trivially $\geq A[n]$

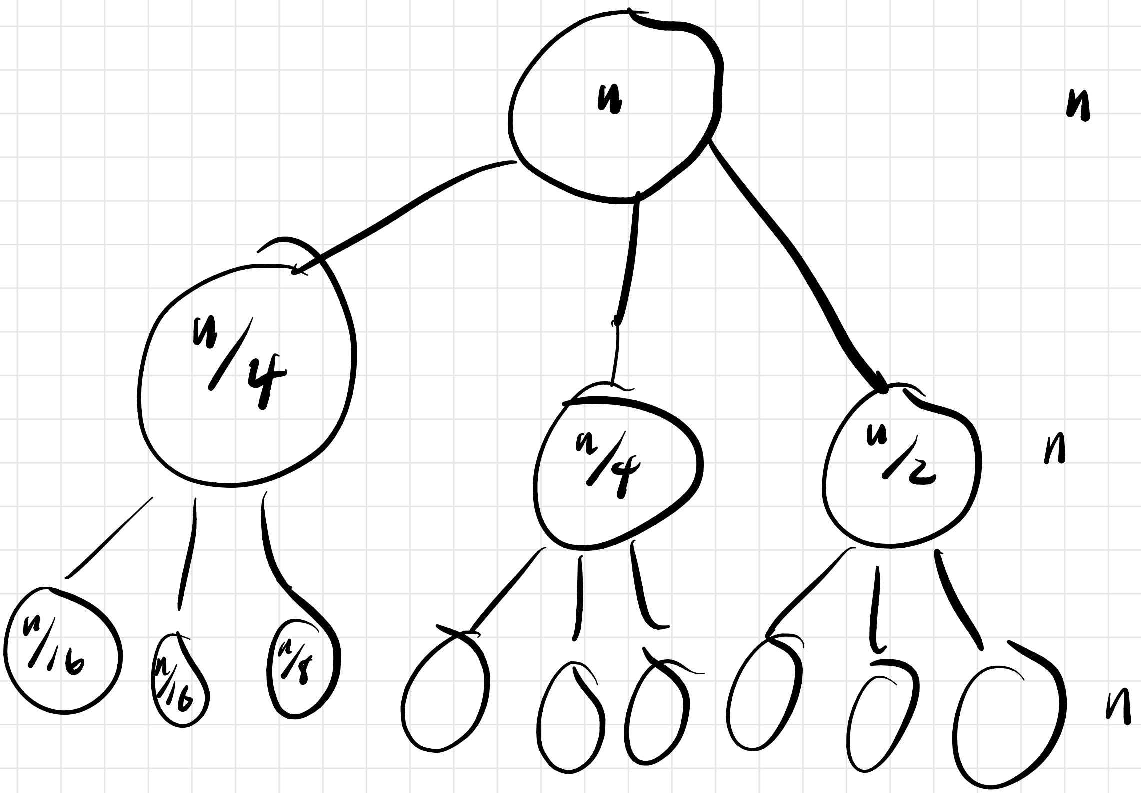
Finally, if $i = 0$, both subarrays are empty + claim is trivial

Theorem: $P(n)$ for every positive integer n.

Proof by induction: Let n be an arbitrary positive integer.
Assume that $P(k)$ is true for every positive integer $k < n$.
There are several cases to consider:

- Suppose n is *. . . blah blah blah . . .*
  Then $P(n)$ is true.
- Suppose n is *. . . blah blah blah . . .*
  The inductive hypothesis implies that *. . . blah blah blah . . .*
  Thus, $P(n)$ is true.

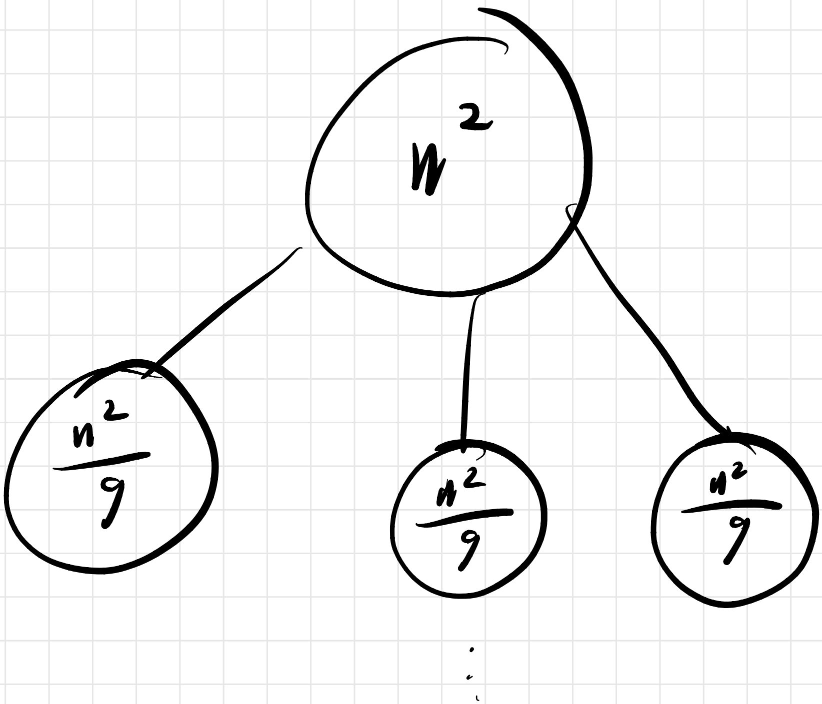
In each case, we conclude that $P(n)$ is true. □



$\le$: $n \cdot \log_2 n = O(n \log n)$

$\ge$: $n \cdot \log_4 n = \Omega(n \log n)$

so $D(n) = \Theta(n \log n)$



n^2

$\frac{n^2}{3}$

$\frac{n^2}{3^i}$

$F(n) = \Theta(n^2)$

$n = F_i + a$

no consecutive gaps?

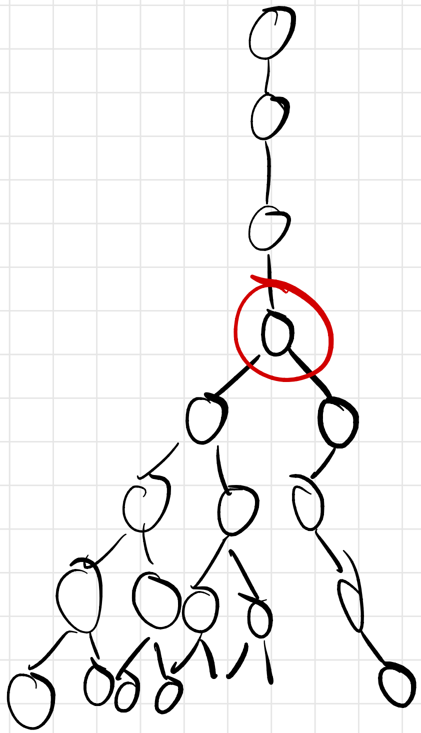
$$\underbrace{F_? + F_? + F_?}_{\text{no consecutive gaps!}}$$

$\log_2 3 \approx 1.5\ldots$

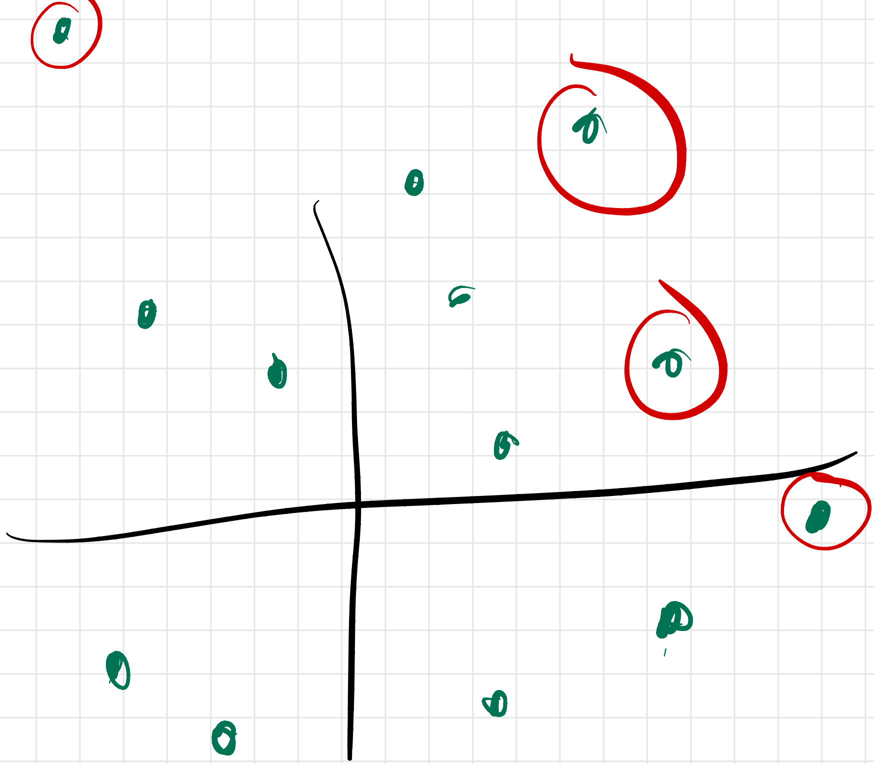
$$n^{\log_2 3} = \omega(n)$$

$$n^{\log_2 3} = O(n^2)$$

$$n^{\log_2 3} \neq \Theta(n^2)$$



$$
\text{CutRod}(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max\limits_{1 \le j \le i} \left[P[j] + \text{CutRod}[i-j] \right] & \text{o.w.} \end{cases}
$$



$P[1..n]$, k

MaxProfit(i, o, k): max profit working days i through n, where we own a share on day i iff o, & we can buy k more times.

MaxProfit(i, o, k) =

$$0 \qquad \text{if } i > n$$

sell today max $\{P[i] + \text{MaxProfit}(i+1, F, k),$
$\text{MaxProfit}(i+1, T, k)\}$ if $i \le n$, o

0 if $i \leq n,\ k = 0,$

$\neg o$

$\max\{-P[i] + \text{MaxProfit}(i+1, T, k-1), \text{MaxProfit}(i+1, F, k)\}$ o.w.

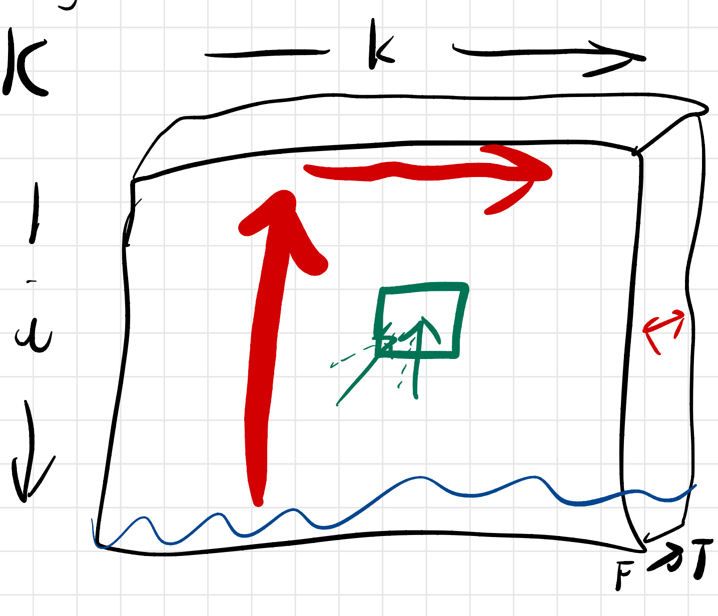
Goal: $\text{MaxProfit}(1, F, K)$

$1 \leq i \leq n+1$

$o \in \{F, T\}$

$0 \leq k \leq K$

so use $\text{MaxProfit}[1..n+1, \{F, T\}, 0..K]$



for $i \leftarrow n+1$ down to 1

  for $k \leftarrow 0$ to K

    for $o \in \{F, T\}$

      MaxProfit[i, o, k] $\leftarrow$ what the recurrance says

time: $O(nK)$

MaxProfit2(i, k): Max profit on days $i \to n$, with k buy ~~lest~~ starting with no share.

$$\text{MaxProfit2}(i,k) = \begin{cases} 0 & \text{if } i>n \text{ or } k=0 \\ \max\left\{\text{MaxProfit2}(i+1,k),\ -P[i] + \max\limits_{i+1 \le j \le n} \left\{ P[j] + \text{MaxProfit}(j+1, k-1) \right\} \right\} \end{cases}$$

time: $O(n^2 k)$

$O(nk)$ subproblems

$O(n)$ time per "

$maxSum(j, x)$

: max sum of a subarray ending at $A[j]$

at most x elements

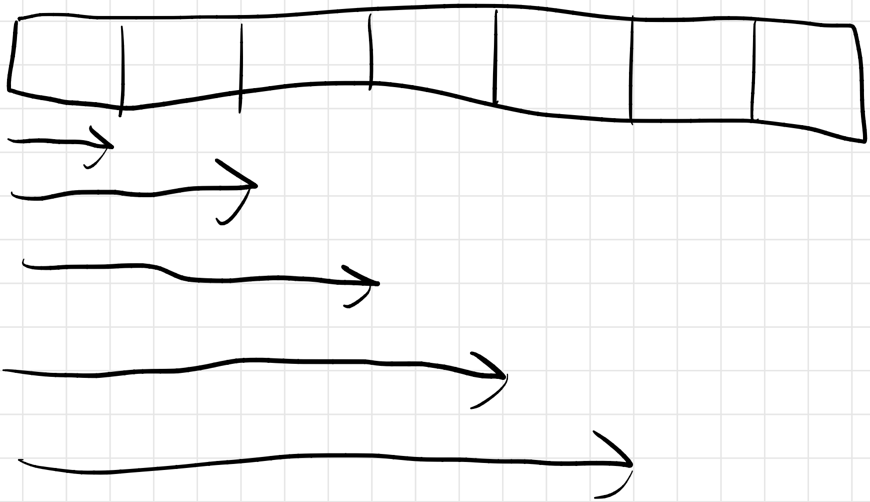
$$maxSum(j, x) = \begin{cases} A[j] + \max\{maxSum(j-1, x-1), 0\} & \text{if } j > 1, x \geq 1 \\ -\infty & \text{if } x = 0, \\ A[j] & \text{if } j = 1, x \geq 1 \end{cases}$$

want to return...

$$\max_{1 \le j \le n} \text{maxSum}(j, X)$$

$1 \le j \le n$
$0 \le x \le X$

time: $O(nX)$



- compute prefix sums in O(n

$$P(i) = P(i-1) + A[i]$$

~~max~~ min-queu :

has push-back(index, value)

pop-front()

~~max~~() min()

q holds prev X indices valued by P[index-1]. Best subarray ending at $A[i] \equiv \max_{j \in q} P[i]-P[j-1] \equiv \min_{j \in q} P[j-1]$

q holds last X indices up to i-1 & an vab

$\max_{} B(j, i-1) \equiv \min P[j-1]$

MaxSum(i, X)

only consider X-1 items up to A[i-1]

if ~~max~~ $i \geq X+1$

q.pop-fro-()

q.push-back(i, P[i-1])

best ← max{best, P[i]-P[q.min()]}

return best

The min-queue.

a) keep a queue q_1

b) store a doubly linked list l

head of l points to min value node in q_1

each node x in l points to min value node y that is <u>behind</u> x in q_1

front ←　　　　　　　　　　　　back

q_1 : 12　2　-5　6　17　8　500　7

l : head → -5 → 6 → ~~8~~ → ~~500~~

7

O(1) min(): return head of ℓ

O(1) pull_front():

if q_1.front = ℓ.head

delete ℓ.head

return q_1.pull_front()

push_back(index, value):

q_1.push_back(index)

while ℓ.tail().value ≥ value

delete ℓ.tail()

add index + value to tail of ℓ

each node in ℓ is deleted at most once across all operations

so over k operations, we spend O(k) time

⇒ O(1) amortized time per ~~pull_front~~ push_back

V: (galaxy, amount spent mod 5)

(amount of small change)

E: $(a, x) \to (v, y)$

iff $y - x \equiv c(av) \bmod 5$

BFS$(s, 0)$

return length of shortest path to $(t, 0)$

Time: $O(n+m)$

$|V| = 5n$

$|E| = 10m$

~~OptExpr(i, j, dir)~~

MinE(i, j): max expr value from ith number to jth

MinE(i, j): min "

use the op. between $A[k]$ & $A[k+1]$

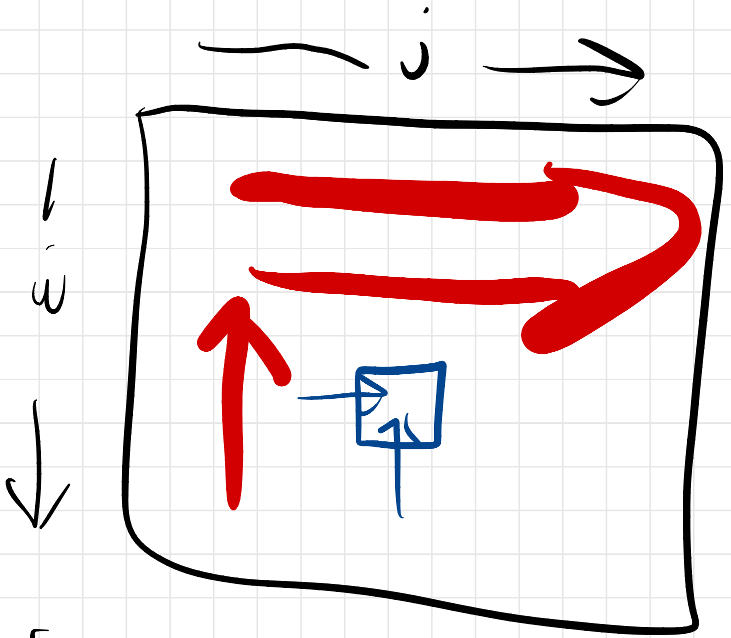
$$\text{MaxE}(i,j) = \begin{cases} A[i] & \text{if } i = j \\ \max\limits_{i \le k < j} \begin{cases} \text{Max}^E(i,k) + \text{Max}^E(k+1,j) \\ \text{Max}^E(i,k) - \text{Min}^E(k+1,j) \end{cases} & \end{cases}$$

k: rightmost integer in leftmost subtree

MinE is analogous (you should write it out)

MinE[1..n, 1..n]   n: # numbers

MaxE[1..n, 1..n]



for j going from 1 to n

  for i ← n to 1

    do min then max

time: $O(n) \cdot 2n^2 = O(n^3)$

$$(1 + 3 - 2) \overset{?}{-} 5 + 1 - 6 + 7$$

?

$$\text{MaxSum}(i,j) = \begin{cases} i\text{th integer} & i = j \\ \max\limits_{i \le k < j} \begin{cases} \text{MaxSum}(i,k) + \text{MaxSum}(k+1,j) & \text{if } (+) \text{ after } j \\ \text{MaxSum}(i,k) - \text{MinSum}(k+1,j) & \text{o.w.} \end{cases} \end{cases}$$

integer just to left of sign

$$\text{MinSum}(i,j) =$$

same, but max $\leftrightarrow$ min

Max $\leftrightarrow$ Min

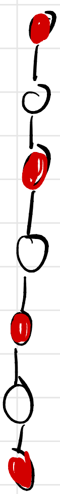
MaxSum(i , j): biggest combination from ith integer to jth

MinSum (i, j) : analogous

~~SSW~~ SSV(v, p): smallest subset size in v's subtree where parent has distance p to nearest ancestor in hypothetical cluster

$$
SSV(v,p) = \begin{cases} 1 + SSV(left(v), 0) + SSV(right(v), 0) & v \text{ is root} \\ \text{same} & p = r \\ \min \begin{cases} \text{same}, \\ SSV(left(v), p+1) + SSV(right(v), p+1) \end{cases} & \text{o.w.} \end{cases}
$$

r = 2



~~r trees?~~

1 tree with an array on each node?

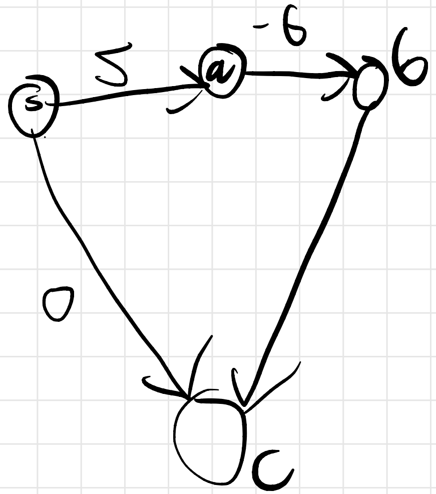
```
BFS(s):
  InitSSSP(s)
  Push(s)
  while the queue is not empty
      u ← Pull()
      for all edges u→v
          if dist(v) > dist(u) + 1          ⟨⟨if u→v is tense⟩⟩
              dist(v) ← dist(u) + 1
              pred(v) ← u                     ⟨⟨relax u→v⟩⟩
              Push(v)
```

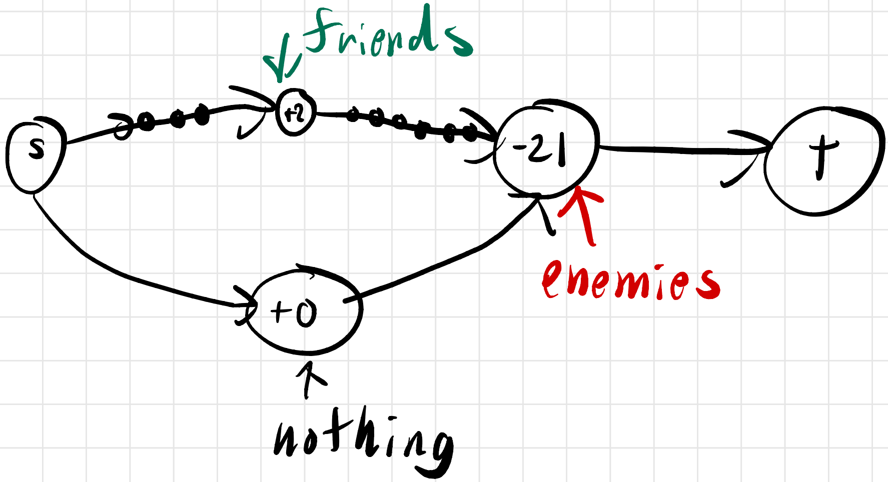
priority queue:
has ExtractMin that does not care about insert order

"min/max queue"
has GetMin but may not support removing the min
Insert only to back
Delete only from front

most implementations do $O(1)$ (amortized) operations

most implementations have $O(\log n)$ ExtractMin





Read about dynamic programming on a DAG.

$G' = (V', E')$

G: $s \xrightarrow{-5} t$

V' : tails of negative edges

heads of negative edges

s, t

E' : $\big((\{s\} \cup \text{heads}) \times (\text{tails} \cup \{t\})\big)$

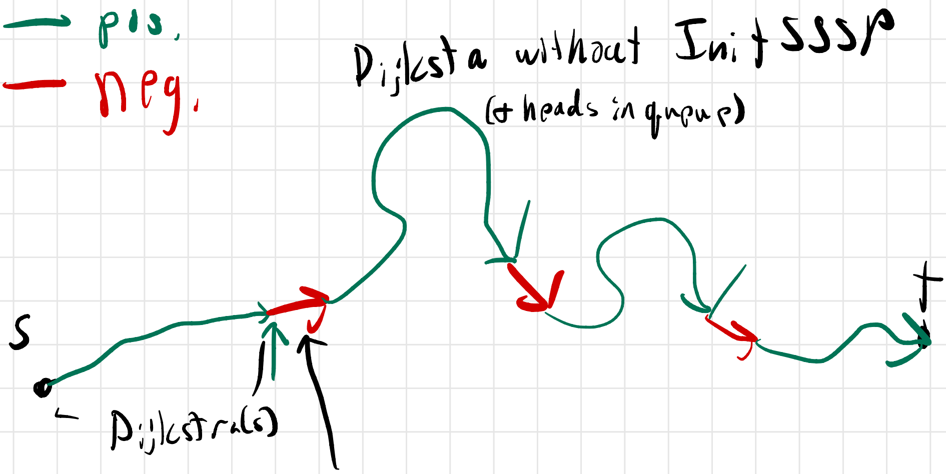
$\cup$ all neg. edges

e', $w'(e') := \begin{cases} w(e) & \text{if } e \text{ is neg.} \\ \text{distance along non-neg edges} & \text{o.w} \end{cases}$

$O(k E \log V)$ to construct G'

BF in $O(V'E') = O(k^3) = O(k^3 E \log V)$

so $O(k E \log V + k^3)$



$O(kE\log V)$

tail $\Rightarrow$ head

Erickson Lemma 8.6

Claim: Let f be an (s,t)-flow + (S,T) be a (s,t)-cut in some flow network $G=(V,E)$ + s,t.

$c: E \to \mathbb{R}_{\geq 0}$.

(f is a max value flow + (S,T) is a min capacity cut)

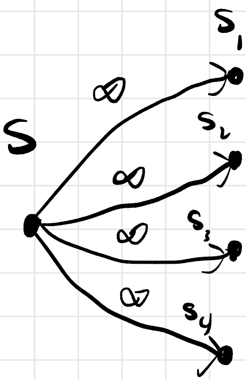
$\iff$

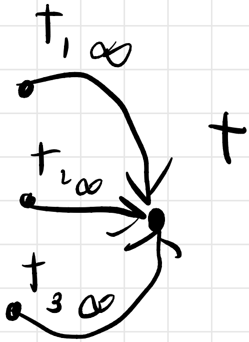
(f saturates all edges from S to T + f avoids all edges from T to S.)

(see Erickson Lemma 10.1)

$$\{ s_1, s_2, \ldots \}$$

$$\{ t_1, t_2, \ldots \}$$





$[0,1)$

x_0

x_1

x_2

x_3

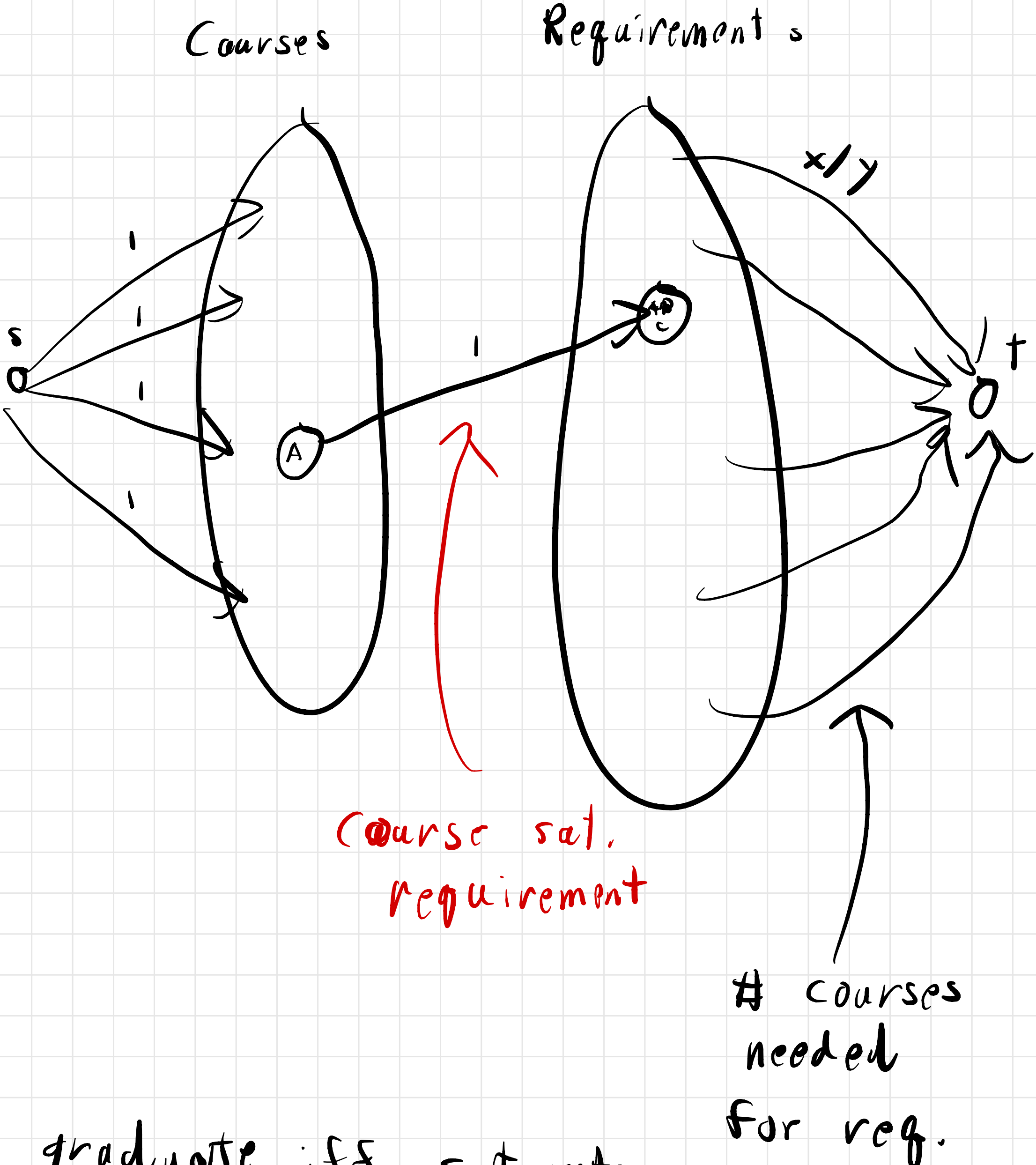
$\vdots$

x

$\vdots$

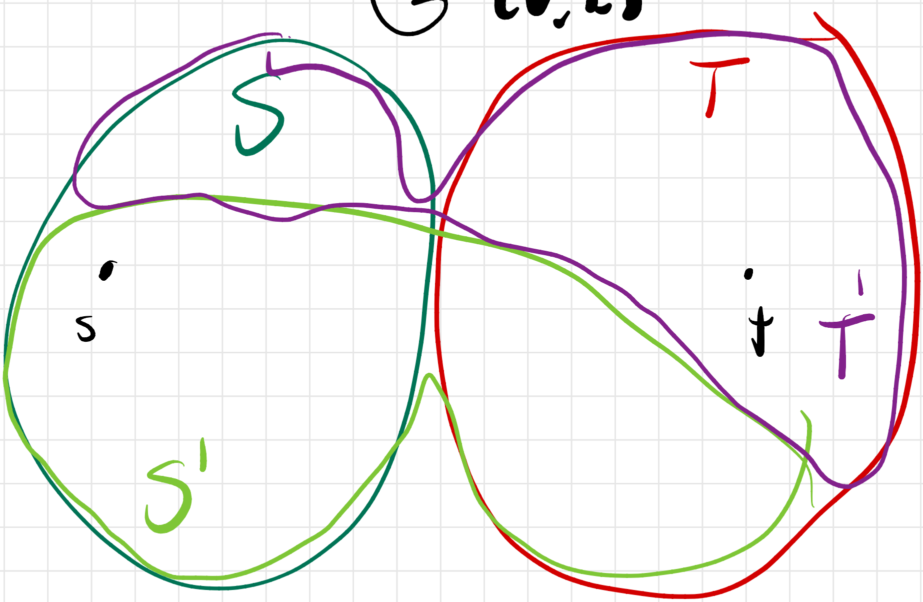
$X :=$ ith bit = the opposite of the ith bit of the ith number in the list

so x is a real number at position j. Its jth bit is...



graduate iff saturate all edges into t

$G = (V, E)$ $\quad T = V \setminus S$



$s \in S \cap S'$

$t \in T \cup T'$

$(S \cap S') \cap (T \cup T') = \emptyset$

$(S \cap S') \cup (T \cup T') = V$

Suppose $u \to v$ leaves $S \cap S'$

$\Rightarrow u \in S$ & $u \in S'$

v is not in both, so $v \in T$ -or- $v \in T'$

$\Rightarrow f^*(u \to v) = c(u \to v)$

To find a min (s,t)-cut given max (s,t)-flow f^*:

intersection of all $\bar{S}$ from min (s,t)-cuts $(\bar{S}, \bar{T})$

$S :=$ reachable from s in G_{f^*}

$T := V \setminus S$.

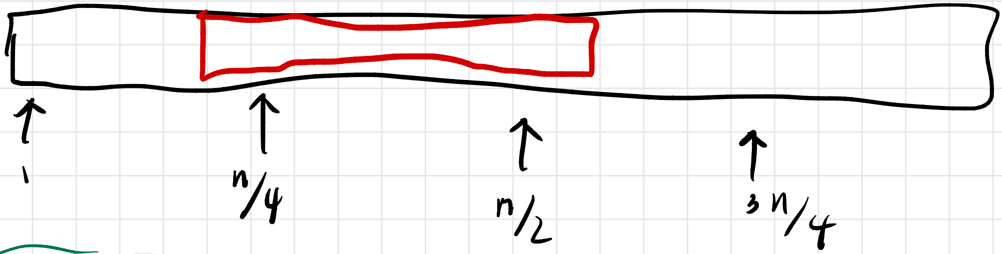
$\exists$ a paths from s to exatly the reachable vertices

(every marked vertex in your favorite search alg from s) (BFS(s) or DFS(s))

$T' :=$ reachable from t in reversal graph (i.e. all vertices that can reach t in G_{f^*})

Return if $T = T'$.

S2019 P1 $(n = 4k)$



Claim: If there are $> n/4$ elements of some equal value, one has a rank in $\{1, n/4, n/2, 3n/4\}$.

So we just need to know the elements of those four ranks

So run Select four times & check if any of the results appears $> n/4$ times.

So $O(n)$ time total.

→ 2019 P3ai:

Find spanning of min total vertex weight.

- return any spanning tree!
  (from BFS, etc.)
  $O(E)$ time

3b: Find an (s,t)-path of min total vertex weight.

(input graph $G = (V, E)$ is undirected + has positive vertex weights $w: V \to \mathbb{R}_{>0}$

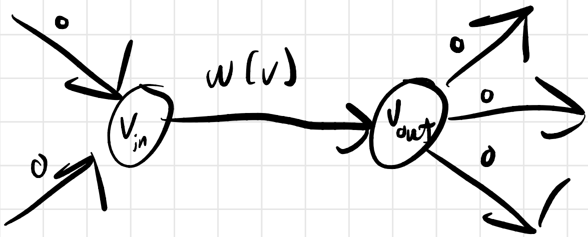
Make directed $G' = (V, E')$.

$\forall\ uv \in E$, add $u \to v$ + $v \to u$ to E'

$w(u \to v) := w(v)\ \forall\ u \to v \in E'$

now Dijkstra.

$$O(E' \log V) = O(E \log V)$$



S 2019 P 2:

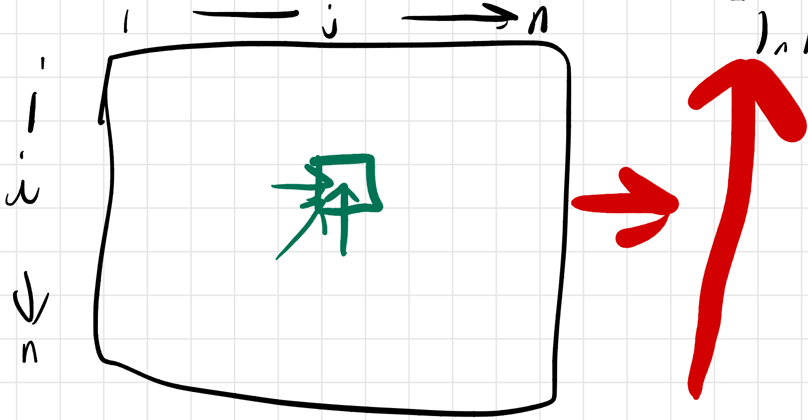
Given $X[1..n]$ of characters.

MaxPalSub(i,j): length of longest subsequence of $X[i..j]$ that is a palindrome.

MaxPalSub(i,j) =

$$\begin{cases} 0 & i > j \\ 1 & \text{if } i = j \\ 2 + \text{MaxPalSub}(i+1, j-1) & \text{if } i < j \text{ + } X[i] = X[j] \\ \max\left\{ \begin{array}{l} \text{MaxPalSub}(i, j-1), \\ \text{MaxPalSub}(i+1, j) \end{array} \right\} & \text{o.w.} \end{cases}$$

b) $1 \le i \le n$ Store in MaxPalSub[1..n,
$1 \le j \le n$ 1..n]



for $i \leftarrow n$ to 1
  for $j \leftarrow 1$ to n
    ...
return MaxPalSub[1, n].

$O(1)$ time per subproblem
$O(n^2)$ subproblems
$\Rightarrow$ $O(n^2)$ time

S 2019 P5:

Given bipartite $G = (L \cup R, E)$

$(\forall\, uv \in E \quad u \in L \,\&\, v \in R)$

$n := |L| + |R|, \quad m := |E|$

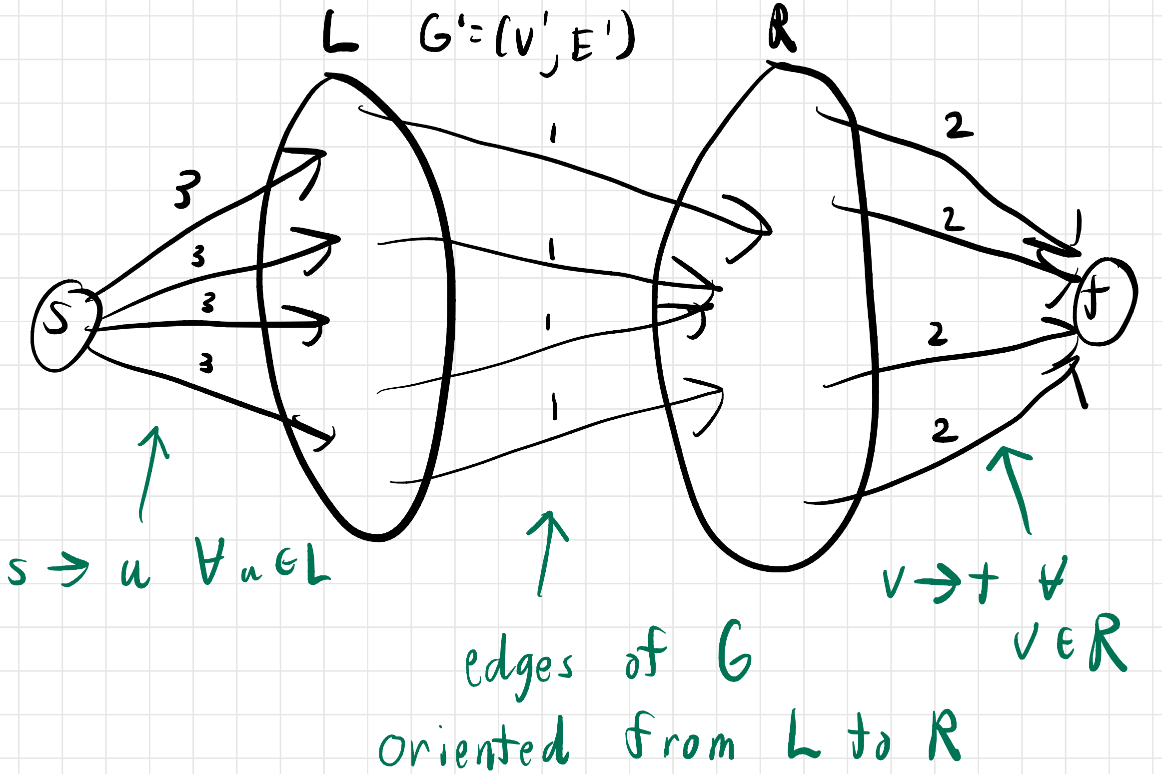
Want largest <u>subset</u> $F \subseteq E$

s.t. every $u \in L$ is incident to

≤ 3 edges of F, &

every $v \in R$ is incident to

≤ 2 edges of F.



$c(u \to v) \leftarrow 1 \quad \forall \ uv \in G$

$c(s \to u) \leftarrow 3 \quad \forall \ u \in L$

$c(v \to t) \leftarrow 2 \quad \forall \ v \in R$

Return value of the max (s,t)-flow. $V' = n+2$, $E' = n+m$

Orlin: $O(V'E') = O(n(n+m))$

With FF: $O(E'|f^*|) = O(n(n+m))$

$|f^*| \leq 2n$