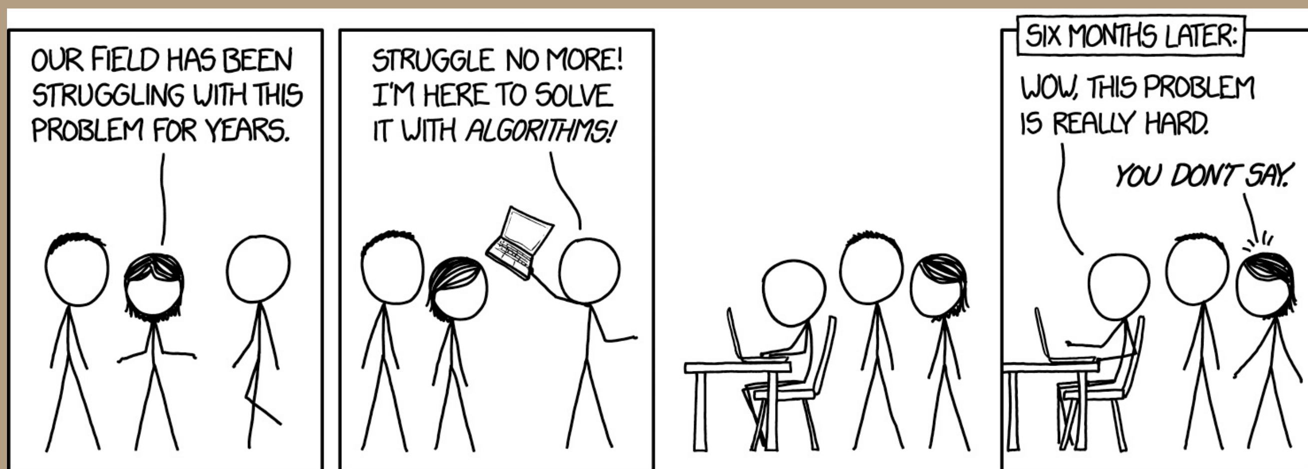
Complexity & Algorithms – Spring '26



Background

Recap

- Any syllabus questions?
- HW0 is posted
- Reading posted for next week
  - → 2 posted
    - discrete math
    - data structures ←
- Class slack ← questions!

Expectations

When I say "give an algorithm", "show how to compute...", etc, what do I mean?

- Pseudocode (+ description)
- Runtime (+ space)
- Proof of correctness

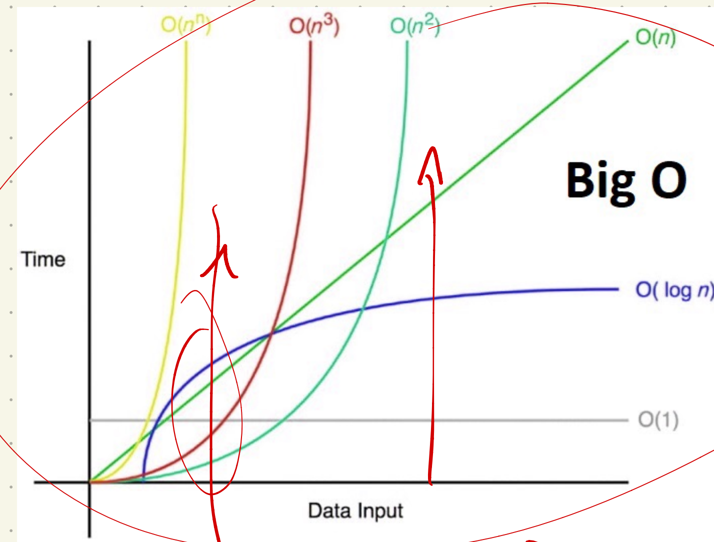
Background

① Big-O & little-o & Θ:

Formally, $f(n) = O(g(n))$

if: $\forall n > N$, $\exists c$ constant

s.t. $f(n) \leq c \cdot g(n)$



$O(1) \ll O(\lg n) \ll O(n)$

↳ Past some target N, f is "dominated by" g

n is $O(\frac{1}{3} \cdot n)$

Why? avoid low level implementation details

Example: $\underbrace{5n^3 - 6}_{f(n)}$ and $\underbrace{14n^3 + 3n^2 + 11}_{g(n)}$

$f(n)$ is $O(g(n))$:

Fix $N = \cancel{7}\ 1$

c

$$5n^3 - 6 \le c\,(14n^3 + 3n^2 + 11)$$

$1 = c$

$$5n^3 - 6 \le 5n^3 - 6 \underbrace{(+9n^3 + 3n^2 + 17)}_{\ge 0}$$

$$= 1 \cdot g(n)$$

$g(n)$ is $O(f(n))$:

set $c = 4$: $c \cdot f(n) = 20n^3 - 24$

littleo: $f(n)$ is $o(g(n))$:

$\forall n > N \ \exists c \geq 0$ s.t.

$f(n) \leq c \cdot g(n)$

$f(n) = \Omega(g(n))$

$6n^2$ is $o(n^2)$

set $c = 7$ Fix $N = 1$.

$6n^2 \leq c \cdot n^2 = 7 \cdot n^2$

big-Theta: $f(n) = \Theta(g(n))$

② logarithms: useful identities!

Find it in your discrete math reference, ie

$$\log_b(xy) = \log_b(x) + \log_b(y)$$

$$\log_b\left(\frac{x}{y}\right) = \log_b(x) - \log_b(y)$$

$$\log_b(x^y) = y\log_b(x)$$

$$\log_b(\sqrt[y]{x}) = \frac{\log_b(x)}{y}$$

Recall Dfn: $\log_a b$ = exponent you take a to in order to get b

$$\log_a b = x \iff a^x = b$$

Identities: how do exponents behave?

$$2^x \cdot 2^y = 2^{x+y} \qquad \log_b(xy) = \log_b x + \log_b y$$

$$\frac{2^x}{2^y} = 2^x 2^{-y} = 2^{x-y}$$

$$(2^x)^y = 2^{xy} = \underbrace{(2^y)}^x$$

$$\underbrace{2^{\log_2 n}} = n$$

"log"

$\ln = \log_e$

$\lg = \log_2$

Another:

$$\log_a b = \left.\frac{\log_x b}{\log_x a}\right] \text{ with any base } x$$

Use it: Show $8\log_{10} n$ is $O(\log_2 n)$:

$$8\log_{10} n = 8\left(\frac{\log_2 n}{\log_2 10}\right)$$

$$= \frac{8}{\log_2 10} \cdot \log_2 n$$

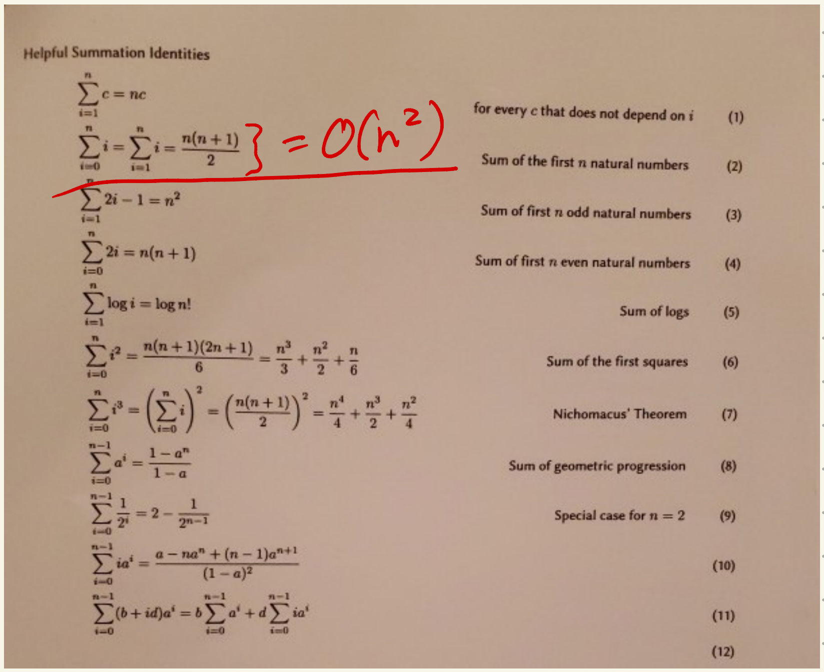
Set $c = \frac{8}{\log_2 10}$ + $N = 10$

then $8\log_{10} n \underset{(=)}{\leq} c \cdot \log_2 n$

③ Summations:

again, your discrete math book has a table. Find it. Love it.

Ex:



Helpful Summation Identities

$\sum_{i=1}^{n} c = nc$	for every c that does not depend on i	(1)
$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \Big\} = O(n^2)$	Sum of the first n natural numbers	(2)
$\sum_{i=1}^{n} 2i - 1 = n^2$	Sum of first n odd natural numbers	(3)
$\sum_{i=0}^{n} 2i = n(n+1)$	Sum of first n even natural numbers	(4)
$\sum_{i=1}^{n} \log i = \log n!$	Sum of logs	(5)
$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$	Sum of the first squares	(6)
$\sum_{i=0}^{n} i^3 = \left(\sum_{i=0}^{n} i\right)^2 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$	Nichomacus' Theorem	(7)
$\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a}$	Sum of geometric progression	(8)
$\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}}$	Special case for $n = 2$	(9)
$\sum_{i=0}^{n-1} ia^i = \frac{a - na^n + (n-1)a^{n+1}}{(1-a)^2}$		(10)
$\sum_{i=0}^{n-1} (b + id)a^i = b\sum_{i=0}^{n-1} a^i + d\sum_{i=0}^{n-1} ia^i$		(11)
		(12)

Notation reminder: $\sum_{i=1}^{n} f(i) = f(1) + f(2) + \dots + f(n)$

(loops!)

Note: $f(i)$ is any function!

$$\sum_{i=1}^{n} 5 = (\overbrace{5+5+\cdots+5}^{n \text{ times}}) = 5n = O(n)$$

$$\sum_{i=1}^{n} i = \underbrace{\sum_{i=1}^{n}\left[\sum_{j=1}^{i} j\right]}_{\text{nested for loops}} = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

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

$$\sum_{i=1}^{n}\sum_{j=1}^{i}(\log_2 n) = \sum_{i=1}^{n} i \cdot \log_2 n$$

$$= \log_2 n \cdot \left[\sum_{i=1}^{n} i\right] = O(n^2 \log_2 n)$$

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + 4 + \cdots + (n-1) + n$$

$$= \frac{n}{2}(n+1)$$

$$\sum_{i=1}^{n} i \cdot \log_2 n = \log_2 n + 2 \cdot \log_2 n + \cdots + n \log_2 n$$

$$= \log_2 n \left[1 + 2 + \cdots + n \right]$$

④ Induction:

There is a template! Fix what you induct on.

Base case: Show true for some small instances

Ind hypothesis: Assume true for all intsances up to some size k

Ind. step: Show true for size $k+1$

Think of this as "automating" a proof:

$$P(1)$$

$$\forall k \geq 1, \quad P(k-1) \Rightarrow P(k)$$

Predicate logic view

Example $\sum_{i=1}^{n} F_i = F_{n+2} - 1$ } : P

Recall F_n : $F_0 = 0$ $\quad F_2 = 1$

$F_1 = 1$ $\quad F_3 = 2$

$F_4 = 3$ ...

$\forall n > 1, \quad F_n = F_{n-1} + F_{n-2}$

Induction on n:

Base case: $n = 1$ LHS: $\sum_{i=1}^{1} F_i = F_1 = 1$ ✓

RHS: $F_{1+2} - 1 = F_3 - 1 = 2 - 1 = 1$

P(1) is true

IH: $\forall k < n$, eq. is true

$$\sum_{i=1}^{k} F_i = F_{k+2} - 1$$

$P(n-1) \Rightarrow P(n)$

IS: Show equality is true for n!

LHS: Consider $\sum_{i=1}^{n} F_i = (F_1 + F_2 + \cdots + F_n)$

$$= F_n + \left(\underbrace{\sum_{i=1}^{n-1} F_i}_{\text{use IH!}}\right)$$

$n-1 = k < n$

$$= \underbrace{F_n + (F_{n+1}}_{\text{by dfn of Fib \#s}} - 1)$$

$$= F_{n+2} - 1$$

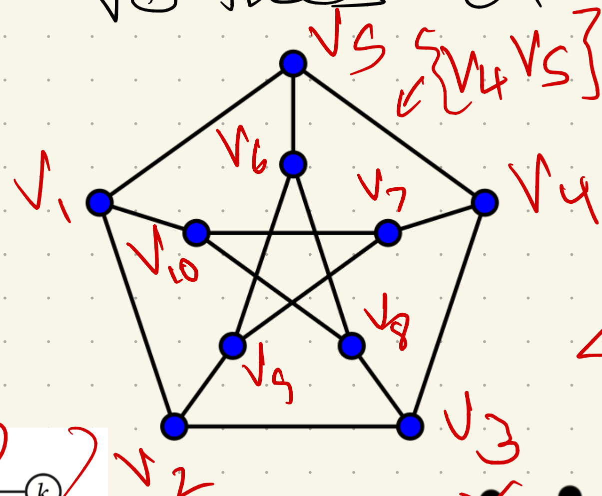
∎

Induction on structures:

Consider graphs: $G = (V, E)$ — vertices, edges

Theorem: In any undirected graph, the number of vertices of odd degree is even.

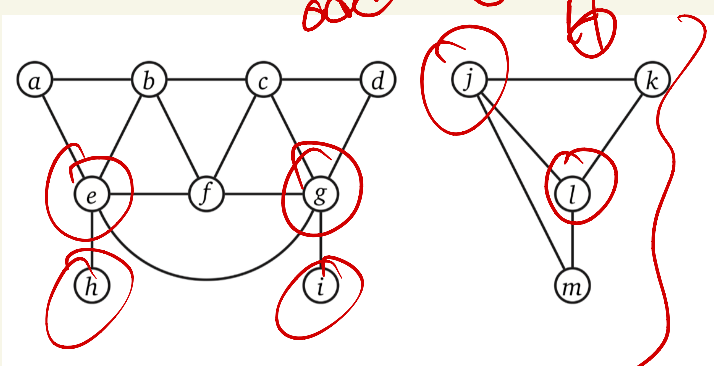
$E = \{(u,v) \mid u, v \in V\}$

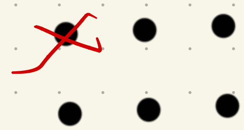


← all vertices have degree = 3

Examples:

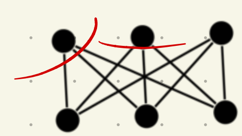
odd degree





Empty graph

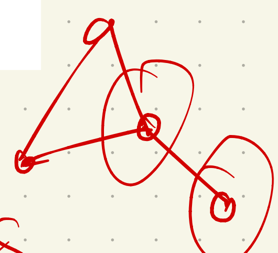
← all 0

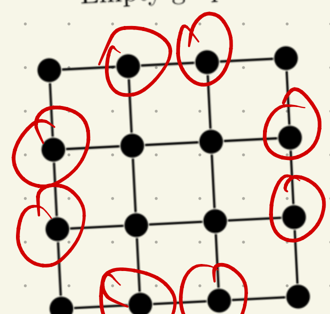


$K_{3,3}$

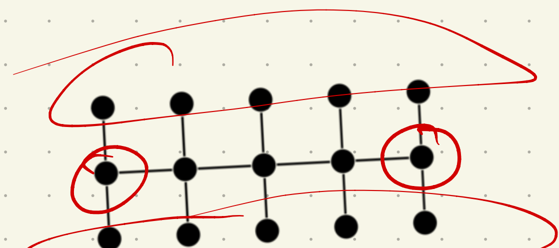
Complete bipartite

6 vertices





Grid



Caterpillar

Note here:

induct on vertices or edges

Proof by induction on the number of <u>edges</u>:

Base case: no edges (<u>any</u> # of vertices)

Any graph with 0 edges has all vertices with $\deg = 0$

$\Rightarrow$ even # of odd degrees $(= 0)$

Inductive hypothesis:

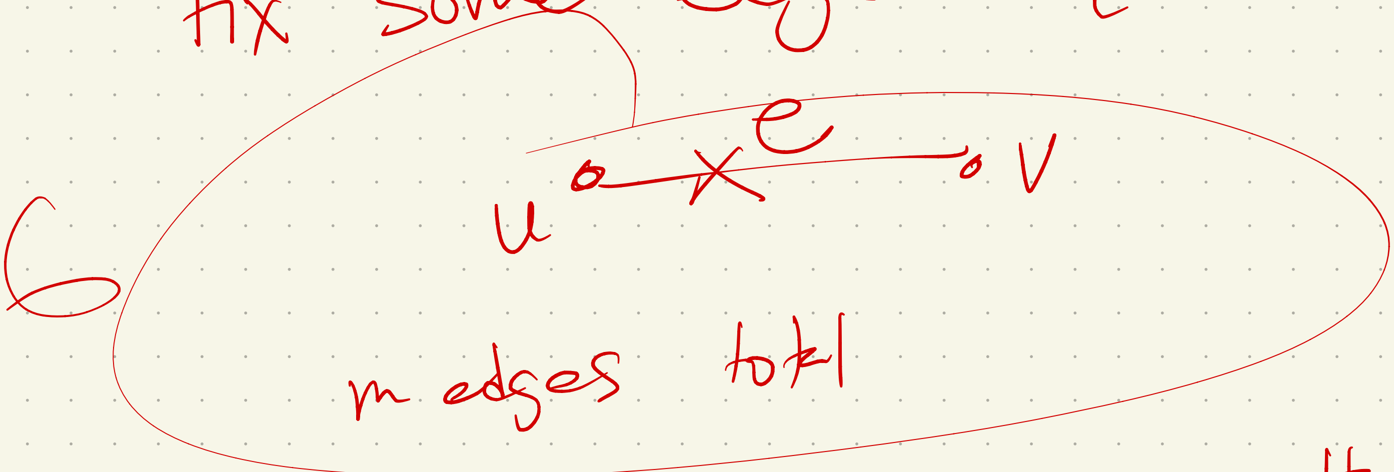
Any graph with $0 \le k < m$ edges has even # of <u>vertices</u> with odd degree.

(any # of vertices)

Inductive Step:

Consider a graph with m edges (Not in base case, so $m \geq 1$)

Fix some edge $e = \{u\ v\}$



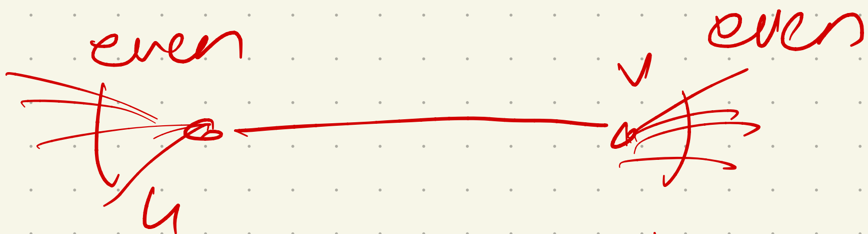
delete e to get G'

By IH, G' has even # of odd degree vertices. $\longrightarrow$ call this X.

What if we re-add e?

Cases:

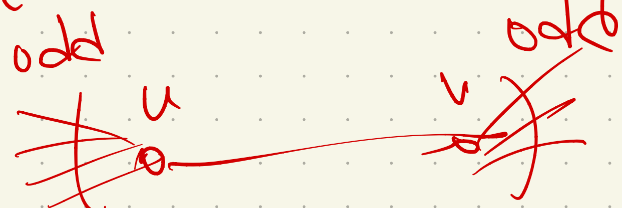
- u and v had even degree in G'



Now, both have odd degree

$\underbrace{x+2}_{x \text{ is even, so is } x+2.}$ odd deg. vertices ✓

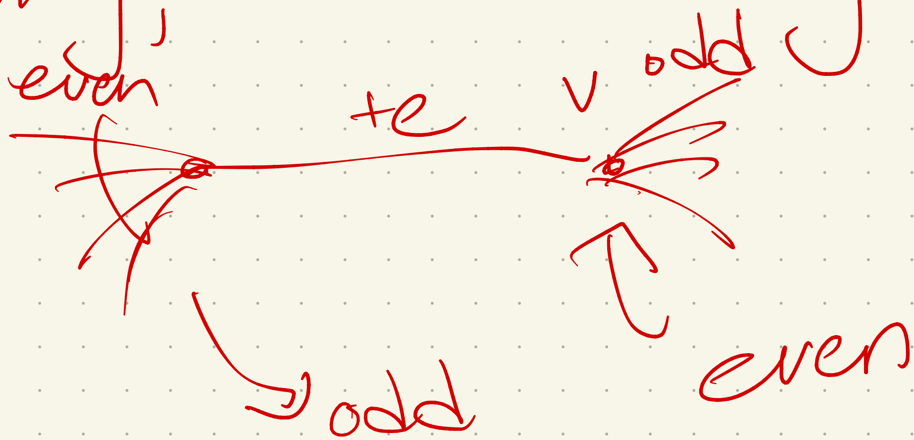
- u & v had odd degree in G'



⟹ now both even

so G has x−2 odd deg. vertices ⟹ still even.

- wlog, u even deg & v odd deg



$\Rightarrow$ still x odd deg. vertices

$\Rightarrow$ Even # of odd degree. ∎

Another: Every rooted binary tree of height h has $\leq 2^{h+1} - 1$ nodes

Recall: $\text{height}(T) = \begin{cases} \\ \\ \end{cases}$

Proof:

5 Pseudo code + runtime:

Discrete math examples (from Rosen textbook)

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1, a_2, \ldots, a_n:$ integers)
$max := a_1$
for $i := 2$ **to** n
  if $max < a_i$ **then** $max := a_i$
return $max\{max$ is the largest element$\}$

This book:

FIBONACCIMULTIPLY($X[0..m-1], Y[0..n-1]$):
  $hold \leftarrow 0$
  for $k \leftarrow 0$ to $n+m-1$
    for all i and j such that $i+j=k$
      $hold \leftarrow hold + X[i] \cdot Y[j]$
    $Z[k] \leftarrow hold \bmod 10$
    $hold \leftarrow \lfloor hold/10 \rfloor$
  return $Z[0..m+n-1]$

Pseudocode conventions here:

Variable assignment:

Boolean comparison:

Arrays: A[0..n-1]
- each element:

Loops:

Pseudocode format:

In a pinch, pretend you're in Python or Ruby → High level & readable.

I realize this is not a "definition" — that is the point!

It's about effective communication.

Next reading: recursion

Most of you indicated you'd seen it before. Topics here:

- Towers of Hanoi
- Merge sort
- Recap of recurrences & "Master theorem"
- Linear time selection
- Multiplication (again)
- Exponentiation

A high level note on recursion:

Recursion really can be simpler & useful!

Often depends upon the language and setup.

Counter-intuitive, but that's often due to lack of practice.

Often considered slower?

Recursion

- If you can solve directly (usually because input is small), do it!
- Otherwise, reduce to simple (usually smaller) instances of the same problem.

Recursion Fairy

- Helps to solidify that "black box" mentality so you don't keep unpacking the next level.

(She's also called the "induction hypothesis".)

Classic example

Our book

```
QuickSort(A[1..n]):
  if (n > 1)
      Choose a pivot element A[p]
      r ← Partition(A, p)
      QuickSort(A[1..r − 1])     ⟨⟨Recurse!⟩⟩
      QuickSort(A[r + 1..n])     ⟨⟨Recurse!⟩⟩
```

```
Partition(A[1..n], p):
  swap A[p] ↔ A[n]
  ℓ ← 0                 ⟨⟨#items < pivot⟩⟩
  for i ← 1 to n − 1
      if A[i] < A[n]
          ℓ ← ℓ + 1
          swap A[ℓ] ↔ A[i]
  swap A[n] ↔ A[ℓ + 1]
  return ℓ + 1
```

Algorithm 1 Quicksort

```
procedure QuickSort(A, p, r)
    if p < r then
        q = Partition(A, p, r)
        QuickSort(A, p, q − 1)
        QuickSort(A, q + 1, r)
    end if
end procedure
procedure Partition(A, p, r)
    x = A[r]
    i = p − 1
    for j = p to r − 1 do
        if A[j] < x then
            i = i + 1
            exchange A[i] with A[j]
        end if
        exchange A[i] with A[r]
    end for
end procedure
```

QuickSort Pseudocode Example

Another version