Running times of algorithms

Bakh Khoussainov

Overall goal:

Find efficient algorithms for computational problems.

But, what does efficient mean?

Attempt 1:

An algorithm is efficient if, when implemented, it runs quickly on real input instances.

This attempt is:

(1) Implementation dependent

(2) Hardware dependent

(3) What is a "real input instance"?

(4) Test case instances dependent?

(5) We do not know the full range of input instances...

Attempt 2

An algorithm is efficient if it achieves qualitatively better worst-case performance than brute-force based algorithm.

The problems with this attempt are:

(1) what does a brute-force algorithm mean?

(2) what do we mean by "qualitatively better".

Proposed definition of efficiency:

An algorithm is efficient if there are constants c, d such that on every input instance of size n the running time of the algorithm is bounded by $c \cdot n^d$ basic computational steps.

Such algorithms are called to have polynomial running time.

This definition of efficiency addresses all the problems posed by the previous attempts.

Asymptotic Upper Bounds.

Let $T(n)$ be a function that represents the worst case running time of some algorithm on inputs of size n.

Let $f(n)$ be another function.

We say $T(n)$ is order $f(n)$,

written $T(n) \in O(f(n))$,

if from some point on

(on natural number line)

$T(n)$ is bounded by a

constant times $f(n)$.

Formally, $T(n) \in O(f(n))$

if there is a constant C and

n_0 such that for all $n \geq n_0$

we have $T(n) \leq C \cdot f(n)$.

Examples:

(1) $T(n) = 4 + 3n + 25n^2$.

Then $T(n) \leq 28n^2$ for all $n \geq 10$. So $T(n) \in O(n^2)$

(2) $T(n) \in O(n^3)$.

Note $T(n) \in O(n^2)$ is better asymtotic bound than $T(n) \in O(n^3)$.

Asymptotic lower bounds.

Say, we proved that $T(n) \in O(n^2)$.

We want to show that this is sharp. This means we want to find a constant ε such that for all sufficiently large number n we have

$$T(n) \geq \varepsilon \cdot n^2.$$

So we say that

$$T(n) \in \Omega(f(n))$$

if there exists a constant ε and number n_0 so that for all $n \geq n_0$ we have

$$T(n) \geq \varepsilon \cdot f(n).$$

Example. $T(n) = 4n^3 + 12n^2 + 7n + 5$

We know $T(n) \in O(n^3)$.

But also $T(n) \geq 4n^3$.

So $T(n) \in \Omega(n^3)$.

If we have proven both

(1) $T(n) \in O(f(n))$

(2) $T(n) \in \Omega(f(n))$

this implies that $f(n)$ is the "right" bound for $T(n)$. Namely, $T(n)$ grows like $f(n)$ within a constant factor. In this case we write $T(n) = \Theta(f(n))$.

Examples:

(1) Let

$$p = 3 + 4n + 17n^2 + 18n^3 + 25n^4.$$

Then $p \in O(n^4)$.

Generally, for

$$p(n) = a_0 + a_1 n + \cdots + a_k n^k$$

we have

$$p(n) \in O(n^k).$$

(2) For p above

$$p(n) \in \Omega(n^k).$$

(3) $\log_a n$ functions.

We know

$$\log_a n = \frac{1}{\log_b a} \cdot \log_b n.$$

So $\log_a(n) = \Theta(\log_b(n))$

These are slow growing functions. In fact, for every $x > 0$,

$$\log_a(n) \in O(n^x).$$

Examples of running times

(1) Finding

$$\text{MAX}\{a_1, a_2, \ldots, a_n\}.$$

$$T(n) = n$$

(2) Selection Sort

$$T(n) \in O(n^2)$$

(3) Merge Sort

$$T(n) \in O(n \log n)$$

(4) Given n points

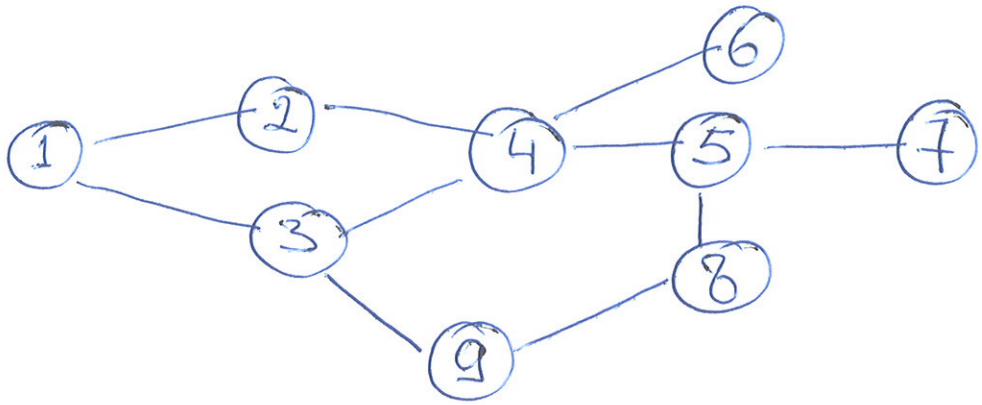
$$P_1, \ldots, P_n$$

find the closest points.

The running time of the brute-force algorithm is bounded by n^2.

(5) Let G be a graph

A subset $X \subseteq G$ is independent if if no to points of X are connected by an edge.



$\{2, 3, 8, 6, 7\}$ are

$\{1, 4, 8\}$ independent.

Let us fix a number $k \geq 1$.

Problem: Input: graph G.

Output: yes, if

G has an independent set of size k.

No, otherwise.

Algorithm:

(1) List all subsets of G of size k:

$S_1, S_2, \ldots, S_t$.

(2) For $i = 1, 2, 3, \ldots, t$

If S_i is independent,

output "yes"; STOP.

(3) Output "no".

Analysis of running time.

For each S_i checking if S_i is independent requires k^2 questions of the type:

Is $(x,y) \in E$?

This is a constant time.

The total number t of subsets of size k in graph G with n vertices is:

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k!} \leq \frac{n^k}{k!}$$

So $T(n) \in O(n^k)$.

(6) Interval Scheduling:

$$T(n) \in O(n \log n)$$

(7) Interval partitioning

$$T(n) \in o(n \log n)$$

$$T(n) \in O(n^2)$$

(8) Lateness minimization

$$T(n) \in O(n \log n)$$

(9) The shortest path problem

$T(n) \in O(n^3)$.

It can be proved

$T(n) \in O(n)$.

(10) MST problem.

$T(n) \in O(n^3)$.

It can be proved

$T(n) \in O(n)$.

A general framework to analizing running times of greedy algorithms:

(1) Analyse time needed to organize input in a desired form (e.g. list intervals in order of deadlines)

(2) Analyze time needed to process the prepared input.

What is the general framework for the analysis of running times of divide and conquer algorithms, (e.g. MergeSort)?

To prove that $T(n) \in O(n \log n)$, we unroll the reccurence relation

$$T(n) \leq 2T(n/2) + cn \leq$$

$$4T(n/4) + 2cn \leq$$

$$8T(n/8) + 3cn \leq \ldots \leq$$

$$\leq 2^j T\left(\frac{n}{2^j}\right) + jcn \leq$$

$$\leq 2^{\log(n)} T(2) + \log(n)\, cn \leq$$

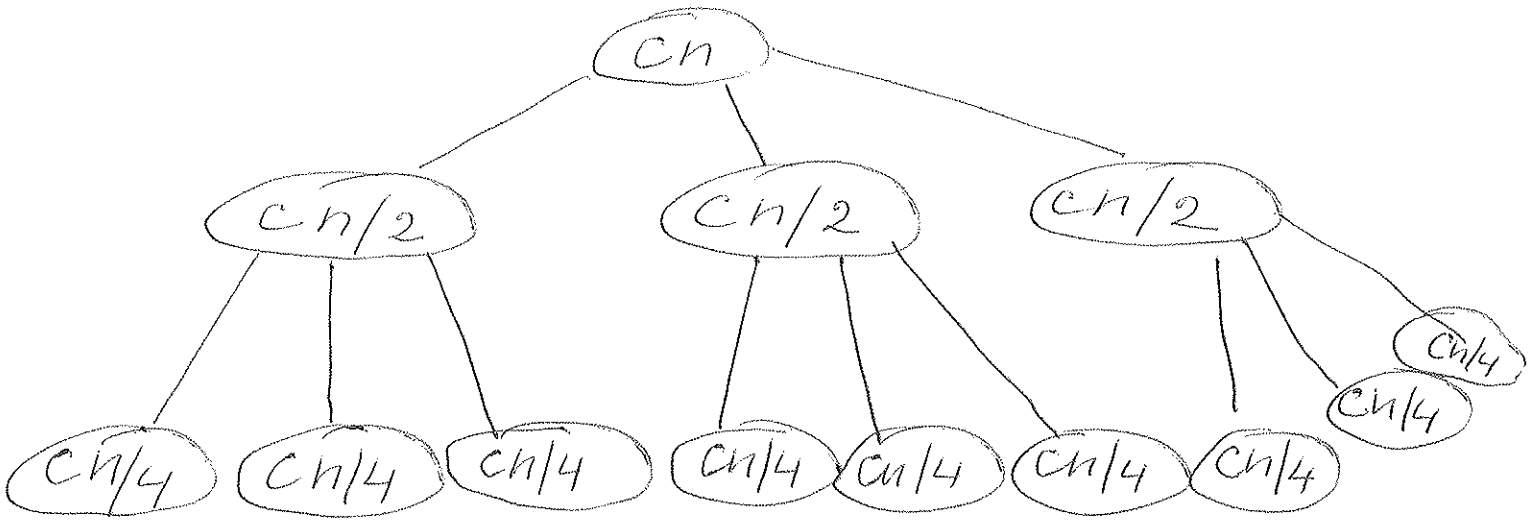
$$\leq (1 + \log(n))\, cn \in O(n \log n).$$

$$T(n) \leq q \cdot T(n/2) + cn$$

$$T(2) \leq c,$$

where $q > 2$.

Picture ($q = 3$)



As above we unroll the reccurence relation

At level j of our recursion we have q^j instances of the problem; all instances are of size $n/2^j$.

So, the total work performed at level j is

$$q^j \cdot \left(\frac{cn}{2^j}\right) = cn\left(\frac{q}{2}\right)^j$$

We sum all these up:

$$T(n) \leq \sum_{j=0}^{\log_2 n} \left(\frac{q}{2}\right)^j cn =$$

$$cn \sum_{j=0}^{\log_2 n} \left(\frac{q}{2}\right)^j .$$

Now, set $r = \frac{q}{2}$, Then

$$\sum_{j=0}^{\log_2 n} r^j = \frac{r^{\log_2 n}}{r-1} .$$

So,

$$T(n) \leq cn \frac{r^{\log_2 n}}{r-1}$$

Now, identity

$$r^{\log_2 n} = n^{\log_2 r} =$$

$$n^{\log_2 q/2} = n^{\log_2 q - 1}$$

So, $T(n) \leq cn \dfrac{r^{\log_2 n}}{r-1} =$

$$= \left(\frac{c}{r-1}\right) n \cdot n^{\log_2 q - 1} = \left(\frac{c}{r-1}\right) n^{\log_2 q}$$

So

$$T(n) \in O\left(n^{\log_2 q}\right).$$

So, the running time is polynomial.