Algorithm design manual

# 1. Introduction to algorithm design

## Robot tour optimization

Algorithm 🡪 always correct result  
 Heuristic 🡪 usually do a good job but without providing any guarantee

Contoh soal: perpindahan titik minimal satu sumbu dan dua sumbu, travelling salesman problem

## Selecting the right job

Contoh soal : aktor dan film optimal  
 Solusi : pilih film selesai tercepat, kemudian hapus film yang berinterferensi

## Reasoning about correctness

Proof 🡪 tools to distinguish correct algorithm to inccorect ones  
 1. Clear, precise statement what to prove  
 2. Set of assumptions  
 3. Reason chain from assumption to proof  
 4. QED(proven) 🡪 demonstration

Correct algorithm 🡪 require careful exposition, and efforts to show both correctness and not incorrectness

### Expressing algorithm

Common forms of algorithmic notation : english, pseudocode, real programming language

“a common mistake students make is using pseudocode to dress up an ill-defined idea to looks formal. Clarity should be the goal”

“The heart of algorithm is idea. If your idea not clearly revealed, then you are using too low-level notation to describe it”

### Problems and properties

Problem specification : [set of allowed input instances] and [required properties of algorithm’s output]

Important and honorable tecnique in algorithm 🡪 narrow the set of allowable instances until there is correct and efficient algorithm.

Common traps in specifying output requirements :  
1. Asking ill defined questions  
Cth: what is the best route? 🡪 what is best definition(parameter)?  
2. Creating compound goals 🡪 too complicated  
Cth : best route 🡪 shortest path from a to b and not using more than twice as many turns as necesarry

### Demonstrating incorrectness

Counter-example 🡪 produce instance that yields incorrect answers

Good counter-example parameter :  
1. Verifiability : [calculate what answer your algorithm will give] and [display a better answer to prove the algorithm didn’t find it]  
2. Simplicity 🡪 Cth: reducing the number of overlapped segments

Counter-example techniques:  
1. Think small 🡪 use simple example on which they fail  
2. Think exhaustively 🡪   
3. Hunt for the weakness 🡪 Cth : if the proposed algorithm is greedy algorithm, think about the way that might tihink it wrong  
4. Go for a tie 🡪 Cth : if using greedy algorithm, make the instance in the sam size 🡪 return something suboptimal answer  
5. Seek extremes 🡪 extreme points

### Induction and recursion

Failure to find counter-example doesnt mean it’s obvious that the algorithm is correct

Mathematical induction (rumus umum penjumlahan n angka) is usually the right way to verify the correctness of a recursive or inremental insertion algorithm

### Summations

Arithmetic progressions and geometric series

## Modelling the problem

Modelling 🡪 art of formulating your application in terms of precisely described, well-understood problem

Must learn to describe problem abstractly, in term of procedures on fundamental structures

### Combinatorial object

Permutations 🡪 arrangements or ordering items (arrangement, tour, order, sequence)  
subsets 🡪 represents selection from a setof items (cluster, collection, committee, group, packaging, selection)  
trees 🡪 represents hierarchical relationship (hierarchy, dominance, ancestor, taxonomy)  
graph 🡪 represesnts arbitrary pairs relationship (network, circuit, web, relationship)  
points 🡪 location in geometric space (sites, positions, data record, locations)  
polygons 🡪 regions in geometric space (shapes, regions, configurations, boundaries)  
strings 🡪 sequence of char or pattern (text, character, pattern, label)

### Recursive object

Learning t think recursively is learning to look for big things that are made from smaller things of exactly the same type of the big things

## War story : Psychic Modelling

The optimal number of ticket to win the lottery [paper Younas and Skienna]

# 2. Algorithm analysis

Tools to compare efficiency of algorithm [the ram model of computation] and [the asymptotic analysis of worst-case complexity]

## The ram model of computation

RAM 🡪 machine-independent algorithm design depends upon hypothetical computer

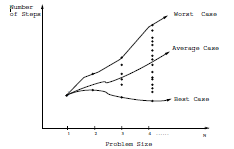
Simple operation (+,-,...) 🡪 one time step  
loops and subroutines 🡪depends in the number of loop iteration ( from 1 to n)  
each memory acces takes only one step. Ram model takes no notice whether the item is in cache or disk

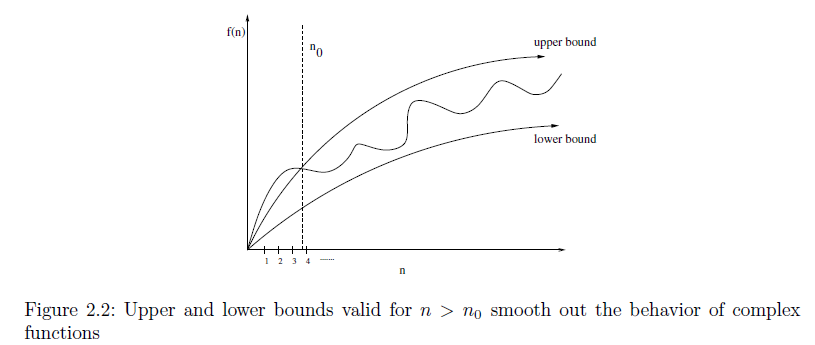
RAM model 🡪 compute how many stepsalgorithm takes on given instances by executing it

### Best, worst, and average-case complexity

Running an algorithm over all possible instances of data can be fed

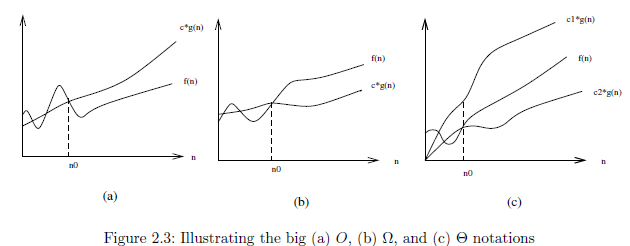
Numerical function : representing time vs problem size



Weakness:  
1. Have too many bumps 🡪 exact time complexity function for any algorithm is liable to be very complicated  
  
2. Require too much detail to specify precisely 🡪 precise answer depends upon uninterseting coding (too detail)

## The big Oh notation

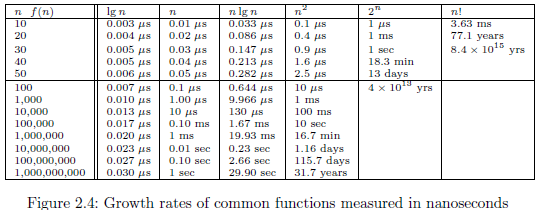
The big oh ignores the difference between multiplicative constant



Dont care about small items, but seek which algorithm proves faster in very large item. Ignore details, focus in big pictures.

3*n*2 *−* 100*n* + 6 = *O*(*n*3), because I choose *c* = 1 and *n*3 *>* 3*n*2 *−* 100*n* + 6 when *n >* 3;  
3*n*2 *−* 100*n* + 6 *\_*= Ω(*n*3), because I choose *c* = 3 and 3*n*2 *−* 100*n* + 6 *< n*3 when *n >* 3;  
3*n*2 *−* 100*n* + 6 *\_*= Θ(*n*3), because only *O* applies;

## Growth rates and dominance relations

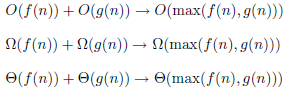


### Dominance relation



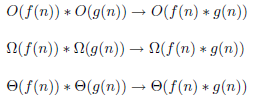
## Working with the big Oh

### Adding function



### Multiplication function

 🡪 if c > 0

If both are increasing, both are important  
 

## Reasoning about efficiency

### Selection sort

Using nested if : two level if  
 so the running time is quadratic : n x n

### Insertion sort

Using while in for function  
 i < n 🡪 always go around n times (outer loop : big oh)  
 so running time is quadratic O(n2)

### String pattern

Find a pattern p in string s 🡪 O(nm)

### Matrix multipication

Multiplication of A[x,y] and B[y,z] 🡪 running time O(xyz) = O(n3)

## Logarithms and their applications

### Logarithm and binary search

Binary search is a good example of an O(log n) algorithm. Cth : find name in phone book

The number of steps the algorihm takes = number of times we can halve n until one name is left 🡪 from n/2 🡪 log2 n

### Logarithms and trees

Cth : height (h) = 2 , nodes (d) = 2 🡪 so there is 4 leaves 🡪 n = 2h or h = log2 n 🡪 n = dh

Binary trees prove fundamental to the design of fast data structures

### Logarithms and bits

## Properties of logarithms

## War story : mystery of the pyramid

## Advanced analysis

# 3. Data Structures

Three fundamental abstract data types : containers, dictionaries, and priority queues

## Contiguous vs linked data structures

Contiguous allocated structures : composed of single slabs of memory and include arrays, matrices, heaps, and hash tables.

Linked data structures : composed of distinct chunks of memory bound together by pointers, and include list, trees, and graph adjacency lists.

### Array

Array is contiguously-allocated data structure 🡪 have some advantages :  
1. Constant-time acces given the index 🡪 index maps directly to particular memory address  
2. Space efficiency 🡪 consists of purely data, no wasted space, fixed size  
3. Memory locality 🡪 to helps exploit the high-speed cache memory

Weakness 🡪 cannot add space in the middle of process

### Pointers and linked structures

Pointers 🡪 connections that hold the pieces of linked structures together. Represents the address of location memory.

List 🡪 simple linked list

Searching 🡪 can be done iteratively or recursively  
search x in the first element of list, if not do the function more (recursive)

Insertion 🡪 insert each new item in the simplest place.  
insertion at the beginning avoid the needs to traverse the list, but require to update pointer

Deletion 🡪 find a pointer to the predecessor of the item to be deleted. After deleted, reset the pointer to the head of the list.

### Comparison

Relative advantages of linked list over static array:  
1. Overflow on linked structures can never occur unless the memory is actually full  
2. Insertions and deletions are simpler than contiguous array  
3. With large records, moving pointer is easier and faster than moving the items themselves

Relative dsadvantages :  
1. Require extra spaces for stroing pointer fields  
2. Do not allow efficient random access to items  
3. Arrays allow better memory locality and cache performance than random pointer jumping

List 🡪 chopping the first element off a linked list and leaves smaller linked list

Array 🡪 stripping the first k elemnt out of n element array gives two smaller arrays

## Stacks and queues

Stacks 🡪 last in first out (LIFO). Simple to implement and very efficient. Right to use when retrieval order doesnt matter. Cth : processing batch jobs. Using push and pop.

Queues 🡪 first in firs out (FIFO). Control waiting time of services. Right to use when order is important

## Dictionaries

Dictionaries 🡪 permits access to data items by content. Primary operations is : search, insert, delete, max, min, predecessor, and sucessor.

Dictionaries avoid details of data structure’s representation and focus atthe task.

Implementation : binary search tree and hash tables

## Binary search trees

Fast search and flexible update. Have fast access to two elements (median) above and below the given node. Using linked list with two pointers per node.

Left subtree < node, right subtree > node.

### Implementing binary search tree

Basic operartions are searching, traversal, insertion, and deletion.

Start at the root node, proceed to the left or right subtree, do recursive as smaller tree.

Traversal in tree : preorder 🡪 root-left-right ; postorder 🡪 left-right-root

### How good are binary search trees?

When using binary search trees, all three dictionaries operation take O(h) time, h is height. Smallest height occur when the tree perfectly balanced, so h = [log n].

It is bad when building trees with insertion, that heights can range from log n to n.

### Balanced search trees

Guarantee the height of tree always to be O(log n) 🡪 adjusting tree after each insertion.

## Priority queues

Provide more flexibility than simple sorting, because they allow new elements to enter system at arbitrary intervals.

Building algorithms araound data structures such as dictionaries and priority queeues leads to both clean structure and good performance.

## War story : stripping triangulations

## Hashing and string

Hash table 🡪 very practical way to maintain a dictionary 🡪 looking up for item in an array takes constant time once you have its index

Hash function 🡪 mathematical function to map keys to integer 🡪 used as index to an array

### Collision resolution

Chaining 🡪 represent the table as an array of m linked list. The *i*th list will contain all the items that hash to the value i. Thus, search, insertion, and deletion reduce to the correspondeing problem in linked list. If the n keys are distributed uniformly in the table, each lis will contain roughly n/m elements, making them constant size when m = n.  
minus : devotes considerable amounts of memory

Open addressing 🡪 hash array as array of elemtns, each initialized as null. On an inserion, check if desired position is empty.

### Efficient string matching via hashing

Most fundamental operation in text string : problem 🡪 substring pattern matching, input 🡪 text string and pattern string, output 🡪 does the string cointain the pattern?

Simple 🡪 check at every position. But runs in O(nm) time.

Rabin-karp 🡪 Linear expected time algorithm for string matching based on hashing. Compute a given hash function on both pattern string and the m-character substring starting from the *i*th position of t. If identical, clearly the resulting hash values must be the same. If different, hash values will almost certainly be different.

### Duplicate detention via hashing

Hashing 🡪 represesnts a large object using a single number.

## Specialized data structures

String and data structures 🡪 character strings are typically represented by arrays of characters, perhaps with a special char to mark the end of the string

Geometric data structures 🡪 consists of collections of data points and region. Regions described by polygons that represented by array of points.

Graph data structures 🡪 represented using either adjacency matrices or adjacency lists

Set data structures 🡪 subsets of items are typically represented using dictionary to support fast membership query

## War story : string ‘em up

## Exercises

# 4. Sorting and searching

## Application of sorting

Clever sorting algorithms run in O(n logn).

Searching 🡪 binary search tests whether an items is in dictionary in O(log n) time, provided the keys all are sorted.

Closest pair 🡪 sort number, so the closest pair must lie nest each other.

Element uniquness 🡪 sort number and do linerar scan through checking all adjacent pairs

Frequency distribution 🡪 sort then sweep from left to right and count

Convex hull 🡪 like rubber banf stretched over the points in the place and released.  
Once points sorted by x cordinate, the points can be inserted from left to right into the hull. The rightmost point always on the boundary, so it will appear in the hull. Adding new rightmost point cause other to be deleted, but can quickly identified because they lie inside the polygon formed by adding the new point.

## Pragmatics of sorting

Consideration in selecting sorting order:  
 1. Increasing or decreasing order?  
 2. Sorting just the key or entire record?  
 3. What should we do with equal key?  
 4. What about non-numerical data?

## Heapsort: fast soring via data structures

### Heaps

Simple and elegant data structure for efficiently supporting the priority queue operations insert and extract-min.

Work by maintaining partial order on the set of elements which is weaker than the sorted order.

Heap-labeled tree is defined to be a binary tree such that the key labeling of each node dominates the key labeling of each of its children.

Heap is a slick data structure that enables us to represent binary trees without using any pointers. Store data as an array of keys and use the position of the keys to implicitly satisfy the role of the pointers.

### Constructing heaps

Heaps canbe constructed incrementally, by inserting each new element into the leftmost open spot in the array 🡪 ensure the desired balanced shape of the heap-labeled tree, bu doesnt maintain the dominance ordering of the keys

The new key might be less than its parent in a min-heap, or greater than its parent in a max-heap.  
Solution 🡪 swap any such dissatisfied element with its parent.

### Extracting the minimum

Heapify 🡪 percolate-down operation that merges two heaps with a new key.

Heapsort 🡪 excahnge the maximum element with the last element and calling heapify repetedly and ives an O(n log n) sorting algorithm.  
it’s an inplace sort 🡪 uses no extra memory over the array containing the elemnts to be sorted

### Faster heap construction

Heaps can constructed even faster by using bubble\_down procedure and some clever analysis.

## War story: give me a ticket on an airplane

## Mergesort: sorting by divide and conquer

Mergesort 🡪 recursive approach to sorting involves partitioning the elements into two gropus, sorting each of the smaller problem recursively, and then interleaving the two sorted lists to totally order the elements

Can also do concatenante them into one list and call heapsort, but that just would destroy all the work spent sorting our component lists. 🡪 use merge two lists

Mergesort do not rely on random access to elements as does heapsort or quicksort.  
Disadvantage is the need for an auxilarry buffer when sorting arrays.  
Mergesort is claasic divide and conquer algorithm.

## Quicksort: sortng by randomization

## Distribution sort: sorting via bucketing

## War story: skiena for the defense

## Binary search and related algorithms

## Divide and conquer

Splits the problem in halves,solves each half, then stitches the pieces back together to form a full solution. Cth : mergesort, fast fourier transform, and strassen’s matrix multiplication