N nested loops make $O(n^n)$ complexity.

The execution time for a simple max search is linear time $O(n)$. It is also the time for a find algorithm.

Example:

We get the first n integers in array except for one of them.

(INSERT FROM NOTES)

We can check the absent element by calculating the sum of the present elements, the sum of the first n element and finding the first missing element that gets the difference to 0.

The time complexity of any operation in a programming language can be different and it depends on the data structure implementation.

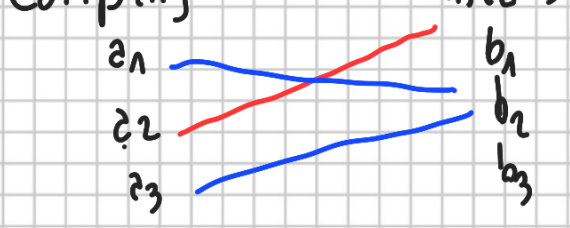
Binary trees work when the data structure is sorted; they work by splitting the search in two halves of data structure and looking in each half.

Time complexity: $O(\log(n))$.

Addition, in general is better than multiplying (Σ / Π), because the latter requires a linear number of bits representation, the former a logarithmic one.

Stable matching

Example with companies and interns:



Likings: (companies)

a_1: $b_2 > b_1 > b_3$

a_2: $b_1 > b_3 > b_2$

a_3: $b_2 > b_1 > b_3$

(interns)

b_1: $a_2 > a_1 > a_3$

b_2: $a_1 > a_3 > a_2$ ← b_2 is wanted by two companies, so he would end up in a_1 basing on its likings.

b_3: $a_3 > a_1 > a_2$

Some definitions:

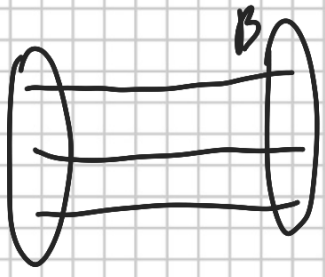
- A and B are 2 disjoint sets of the same cardinality n
($A \cap B = \emptyset$ and $|A| = |B| = n$)

- A perfect matching between A and B is a pairing of the elements of A with the elements of B. In mathematical terms:

$M \subseteq \{\{a, b\} \mid a \in A, b \in B\}$

It is perfect $\Leftrightarrow \forall a \in A \ \exists$ only one b s.t. $\{a, b\} \in M$ and viceversa

Graphically:



There are n! perfect matchings (consider excluded possibilities)
Moreover, it creates a bijective function.

An example of perfect matching:

$M = \{\{a_1, b_1\}, \{a_2, b_3\}, \{a_3, b_2\}\}$

- Stable matching:

Let A, B be two sets s.t. $|A| = |B| = n$ and $A \cap B = \emptyset$
Suppose that each $a \in A$ has a preference list in B, and $b \in B$ has a preference list in A.

For example with 2 hospitals (A) and 2 doctors (B):

$a_1: b_1 > b_2$ | $b_1: a_1 > a_2$
$a_2: b_1 > b_2$ | $b_2: a_1 > a_2$

Here, both a_1 and b_1 would like to switch so they both break up to get with the

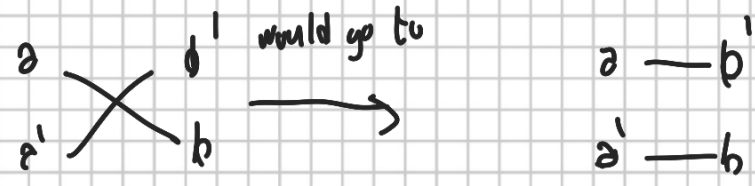
We have two possible matchings:

$a_1 — b_1$ → Here, a_1 and b_1 want break up for the likings of a_2 and b_2. This is the stable matching.
$a_2 — b_2$

$\leftarrow$ $a_1 \times b_2$
$a_2 \times b_1$

reciprocal favourites.

So, the definition of stable matching is:
Two distinct pairs $\{a, b\}, \{a', b'\} \in M$ exhibit an instability if:
- a prefers b' to b, and
- b' prefers a to a'



So, a stable matching is any of the (many!) matchings that contains no instabilities.

Now, we will look at the possibility of the presence, and how to find immediately, at least a stable matching with a single algo.

Gale Shapley():
- Initially all elements are Free.
- While there is some $a_i \in A$ that hasn't yet proposed to each $b_i \in B$:
  - Let a_i be a free element of A.
  - Let $B' \subseteq B$ be the set of elements of B that a_i has not proposed to yet.
  - Let $b_j \in B'$ be the element of B' that ranks highest in a_i's pref. list.
  - If b_j is Free:
    - Match a_i and b_j // [a_i and b_j get engaged, for now]
    - So, a_i and b_j are not Free anymore.
  - If b_j is not Free, and matched to $a_k \in A$:
    - if b_j prefers a_k to a_i:
      - a_i remains free and a_k and b_j remains engaged
    - if b_j prefers a_i to a_k:
      - b_j breaks up with a_k and matches with a_i, and a_k gets free

The time complexity is $O(n^2)$ because there are 2 groups of n that need to be matched. This can be seen in the while loop.

Let's study this algo through some lemmas: (A→male side B→female)

Lemma 1: each $b \in B$ remains matched from the first time she gets a proposal until the end of the execution of the algo. Moreover the partners of b get better from her perspective over time.

Note that this lemma does not guarantee that b gets proposed at all.

Proof: When b gets her first proposal, she accepts it and becomes engaged. She could get other proposals. If she accepts one such subsequent proposal, she'll remained matched with someone she likes better; otherwise, she keeps her partner.

□

Lemma 2: For each $a \in A$, the sequence of proposals of a gets worse, from his perspective, over time.

Proof: By algo's definition.

Theorem 1: The algo ends after at most n^2 iterations.

Proof: Each $a_i \in A$ can propose to at most $|B|=n$ people from B. In each iteration of the loop, an a_i proposes to some b_j that he had not proposed to earlier.
Thus, there can be at most $|A| \cdot |B| = n^2$ iterations.

□

Once proven the algorithms ends, let's prove it produces a perfect stable matching.

Lemma 3: If $a \in A$ is free at some point in the execution of the algo, then there exists at least one $b \in B$ to which a has not proposed yet.

Proof: A contradiction proof.
Suppose that at some point, $\exists a^* \in A$ which is free and that a^* has proposed to each $b \in B$.

By lemma 1, each $b \in B$ is engaged from the first proposal until

the end. Thus, for a^* to be free after $n=|B|$ it must be that at his nth (last) proposal, each $b \in B$ is engaged.
But given $|A|=|B|=n$, for each $b \in B$ to be engaged, this must be true also for A, so a^* cannot exist.

□

Lemma 4: The algorithm outputs a perfect matching.

Proof: The set of engaged pairs always forms a matching (if a makes a proposal, then he is free, when b accepts a proposal, she is either free, or forealls up the previous matching).

Note that matching's neces. 1-1

Suppose, that, in the end, $a \in A$ is free. Then, a has proposed to each $b \in B$. This contradicts lemma 3.

Given that $|A|=|B|=n$, no $b \in B$ can end up free, either. Thus the algorithm returns a perfect matching.

□

TODAY IT'S THURSDAY

Theorem 2: The algorithm outputs a stable matching.

Proof: By lemma 4, the matching M is perfect.

Proof by contradiction:
Suppose that $\{a_i, b_j\}, \{a_k, b_e\} \in M$ and that it is unstable. Then,

① a_i likes $b_e > b_j$
② b_e likes $a_i > a_k$

By the algo's, a_i's last proposal was to b_j.
We consider two cases:

- a_i proposes to b_e before b_j. Then since b_e ended up with a_k, ① entails that b_e prefers a_k to a_i.
  But this contradicts ②.
- a_i did not propose to b_e before b_j (it never proposed, since b_j is a_i's last proposal).
  Consequently, since the order of proposals is determined by the preference list, a_i likes $b_j > b_e$, and this is a

contradiction with ①.

□

The algorithm will always choose the same stable matching.