Algorithms: Survey of Common Running Times

Why It Matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Constant time

Constant time. Running time is $O(1)$.

bounded by a constant, which does not depend on input size n

Examples.

- Conditional branch.
- Arithmetic/logic operation.
- Declare/initialize a variable.
- Follow a link in a linked list.
- Access element i in an array.
- Compare/exchange two elements in an array.
- ...

Linear Time: O(n)

Linear time. Running time is proportional to input size.

Computing the maximum. Compute maximum of n numbers $a_1, \ldots, a_n$.

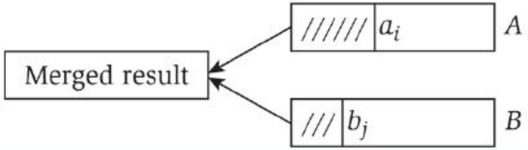
```
max ← a1
for i = 2 to n {
   if (ai > max)
      max ← ai
}
```

Linear time

Linear time. Running time is $O(n)$.

Merge two sorted lists. Combine two sorted linked lists $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$ into a sorted whole.

$O(n)$ algorithm. Merge in mergesort.



$i \leftarrow 1$; $j \leftarrow 1$.

WHILE (both lists are nonempty)

  IF ($a_i \leq b_j$) append a_i to output list and increment i.

  ELSE append b_j to output list and increment j.

Append remaining elements from nonempty list to output list.

Logarithmic time

Logarithmic time. Running time is $O(\log n)$.

Search in a sorted array. Given a sorted array A of n distinct integers and an integer x, find index of x in array.

$O(\log n)$ algorithm. Binary search.

remaining elements

- Invariant: If x is in the array, then x is in $A[lo .. hi]$.
- After k iterations of WHILE loop, $(hi - lo + 1) \le n / 2^k \Rightarrow k \le 1 + \log_2 n$.

```
lo ← 1; hi ← n.
WHILE (lo ≤ hi)
   mid ← ⌊(lo + hi) / 2⌋.
   IF      (x < A[mid])  hi ← mid − 1.
   ELSE IF (x > A[mid])  lo ← mid + 1.
   ELSE RETURN mid.
RETURN −1.
```

Linearithmic time

Linearithmic time. Running time is $O(n \log n)$.

Sorting. Given an array of n elements, rearrange them in ascending order.

$O(n \log n)$ algorithm. Mergesort.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

O(n log n) Time

O(n log n) time. Arises in divide-and-conquer algorithms.

also referred to as linearithmic time

Sorting. Mergesort and heapsort are sorting algorithms that perform O(n log n) comparisons.

Largest empty interval. Given n time-stamps $x_1, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?

O(n log n) solution. Sort the time-stamps. Scan the sorted list in order, identifying the maximum gap between successive time-stamps.

Quadratic time

Quadratic time. Running time is $O(n^2)$.

Closest pair of points. Given a list of n points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is closest to each other.

$O(n^2)$ algorithm. Enumerate all pairs of points (with $i < j$).

```
min ← ∞.
FOR i = 1 TO n
   FOR j = i + 1 TO n
      d ← (x_i − x_j)^2 + (y_i − y_j)^2.
      IF (d < min)
         min ← d.
```

Remark. $\Omega(n^2)$ seems inevitable, but this is just an illusion. [see §5.4]

Cubic Time: $O(n^3)$

Cubic time. Enumerate all triples of elements.

Set disjointness. Given n sets $S_1, \ldots, S_n$ each of which is a subset of 1, 2, …, n, is there some pair of these which are disjoint?

$O(n^3)$ solution. For each pairs of sets, determine if they are disjoint.

```
foreach set Si {
   foreach other set Sj {
      foreach element p of Si {
         determine whether p also belongs to Sj
      }
      if (no element of Si belongs to Sj)
         report that Si and Sj are disjoint
   }
}
```

Cubic time

Cubic time. Running time is $O(n^3)$.

3-SUM. Given an array of n distinct integers, find three that sum to 0.

$O(n^3)$ algorithm. Enumerate all triples (with $i < j < k$).

```
FOR i = 1 TO n
  FOR j = i + 1 TO n
    FOR k = j + 1 TO n
    IF (a_i + a_j + a_k = 0)
      RETURN (a_i, a_j, a_k).
```

Remark. $\Omega(n^3)$ seems inevitable, but $O(n^2)$ is not hard. [see next slide]

Polynomial time

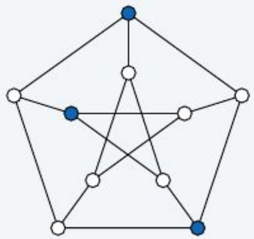
Polynomial time. Running time is $O(n^k)$ for some constant $k > 0$.

Independent set of size k. Given a graph, find k nodes such that no two are joined by an edge.

k is a constant

$O(n^k)$ algorithm. Enumerate all subsets of k nodes.

```
FOREACH subset S of k nodes:
    Check whether S is an independent set.
    IF (S is an independent set)
        RETURN S.
```



independent set of size 3

- Check whether S is an independent set of size k takes $O(k^2)$ time.
- Number of k-element subsets = $\binom{n}{k} = \frac{n(n-1)(n-2) \times \cdots \times (n-k+1)}{k(k-1)(k-2) \times \cdots \times 1} \leq \frac{n^k}{k!}$
- $O(k^2 \, n^k / k!) = O(n^k)$.

poly-time for $k = 17$, but not practical

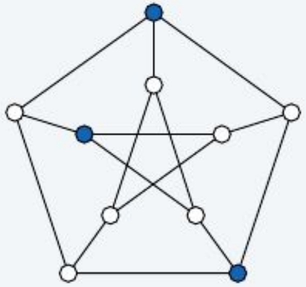
Exponential time

Exponential time. Running time is $O(2^{n^k})$ for some constant $k > 0$.

Independent set. Given a graph, find independent set of max cardinality.

$O(n^2 2^n)$ algorithm. Enumerate all subsets of n elements.

$S^* \leftarrow \varnothing$.
FOREACH subset S of n nodes:
  Check whether S is an independent set.
  IF (S is an independent set and $|S| > |S^*|$)
    $S^* \leftarrow S$.
RETURN S^*.



independent set of max cardinality

Exponential time

Exponential time. Running time is $O(2^{n^k})$ for some constant $k > 0$.

Euclidean TSP. Given n points in the plane, find a tour of minimum length.

$O(n \times n!)$ algorithm. Enumerate all permutations of length n.

$\pi^* \leftarrow \varnothing$.

FOREACH permutation π of n points:

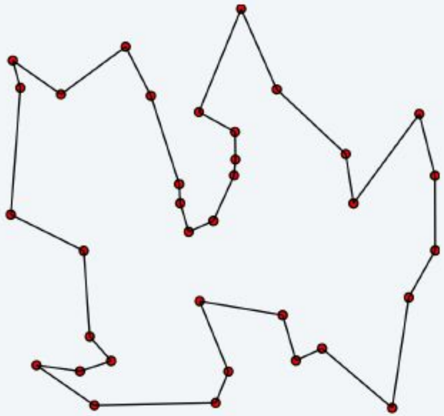
  Compute length of tour corresponding to π.

  IF (length(π) < length(π^*))

    $\pi^* \leftarrow \pi$.

RETURN π^*.

for simplicity, we'll assume Euclidean distances are rounded to nearest integer (to avoid issues with infinite precision)



Why It Matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Suggested Reading

- Algorithm Design by Jon Kleinberg, Eva Tardos
  - Chapter 2
    - Section: 2.4