Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281

ผู้สอน: ตอน 1 ผศ. เบญจมาศ ปัญญางาม เรียน ห้อง 100
ตอน 2 ผศ. ดร. จักริน ชวชาติ เรียน ห้อง 209

บทที่ 3
Survey of common running time

https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/02AlgorithmAnalysis.pdf

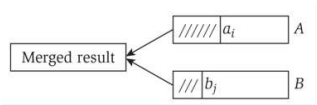
Constant time

- Constant time. Running time is *O(1).*
- 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
  - ...

bounded by a constant, Which does not depend on input size *n*.

Linear time

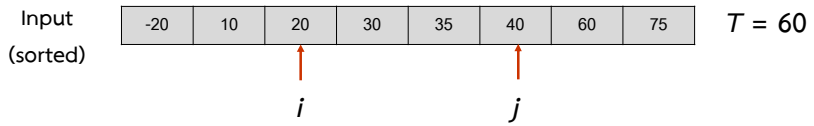
- Linear time. Running time is *O(n).*
- Merge two sorted lists. Combine two sorted linked list $A = a_1,a_2,\ldots,a_n$ and $B = b_1,b_2,\ldots,b_n$ into a sorted whole.
- O(n) algorithm. Merge in mergesort.



i=1;j=1
While (both lists are nonempty)
  if($a_i \le b_j$) append ai to output list and increment i.
  else append bj to output list and increment j.
Append remaining elements from nonempty list to output list.

Target Sum

- Target-Sum. Given a sorted array of *n* distinct integers and an integer *T*, find two that sum to exactly *T* ?



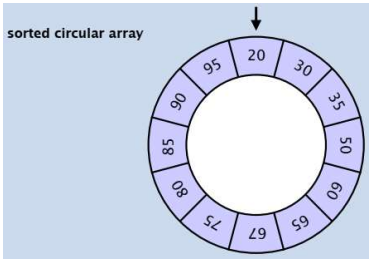
Logarithmic time

- 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.
  - Invariant: if x is in the array, then x is in A[lo .. hi]
  - After k iterations of while loop, (hi-lo+1) < n/ 2k -> k< 1 + log 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
```


Homework 1

- Search in sorted rotated array. Given a rotated sorted array of n distinct integers and an element x, determine if x is in the array



Sorted rotated array

80	85	90	95	20	30	35	50	60	65	67	75

- Find an algorithm to solve this problem

Linearithmic time

- Linearithmic time. Running time is O(nlog n).
- Sorting. Given an array of n elements, rearrange them in ascending order.
- O(nlog 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

Homework 2

- Largest empty interval.
- Given *n* timestamps $x_1, x_2, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval when no copies of file arrive?

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), … (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 = infinity
for i =1 to n
        for j = 1 to n
                d = (xi-xj)2+(yi-yj)2
                if(d < min)
                        min = d
```

- Can improve

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(ai+aj+ak = 0)
                return (ai,aj,ak)
```


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 and edge.
- $O(n^k)$ algorithm. Enumerate all subset of k nodes.

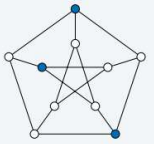
```
Foreach subset S of k nodes: i =1 to n
    check whether S is an independent set
    if (S is an independent set)
        return S
```

- Check whether S is an independent set of size k takes $O(k^2)$ time
- Number of k-element subset = $\binom{n}{k} < \frac{n^k}{k!}$
- $O\left(\frac{k^2 n^k}{k!}\right) = O(n^k)$

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 subset of n elements.

$S^* \leftarrow \emptyset$.
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(nxn!)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)