Sorting:

- WAP: In: Array A of Integers
  Out: A sorted array in ascending ↑

Ideas : (i) Compare pairwise and swap
(Insertion sort)

⊳ steps $\approx \sum_{i=1}^{n} i \simeq n^2/2$

(ii) Divide & conquer (= Merge)
: Divide into halves, sort each, merge two.

⊳ steps $\simeq t(n) = 2 \cdot t(n/2) + O(n)$
$\simeq$ $\boxed{n \cdot \log n \text{ faster than } n^2}$

Q. Which is better?

let $n = 10^7$

(i) $n^2 = 10^{14} \times 1ns$

$= 10^5 s > 28 ms$

(ii) $n \log n = 10^7 \times 24 \times 1ns < 1s$

DATA STRUCTURES (ds)

- Info. stored in a formatted way.
- array is a ds [iterative help] eg: iFib(n)
  - sorted array is an even better ds. why?

Ex: Phone directory / Dictionary.

- suppose it has $n = 10^8$ words

Made with Goodnotes

WAP- $\begin{cases} \underline{\text{In}}: \text{ Dict } A \text{ \& string } s \\ \text{Out: Yes iff } s \in A \end{cases}$

<u>Ideas</u>:

(i) Sequentially: search s in A

Dstep $\approx n \simeq 10^8 \times 10^{-9}$ s $\overset{\sim}{=}$ 0.1s / query

(ii) Binary search for s in <u>sorted</u> A.

$1 \qquad \underset{n/4}{\uparrow} \quad \underset{\frac{3n}{8}}{\uparrow} \quad \underset{n/2}{\uparrow} \qquad n$

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \; \ldots \; \boxed{\log n}$

$D\#$ steps $\simeq \log n \simeq 24 \times 1$ns

$= 30$ns / query.

⇒ DS stores data in the memory in a way that each subsequent operation (query / update) is fast

&

a ds-invariant is maintained.

- sorting & ds are used heavily in all applications.

Problem: Range-minima

WAP - { In : Array A of int.
Out : A fast ds to answer queries of type: Rng-min(i, j)

[finds the min of A[i], A[i+1] ... A[j]].

i=2 j=4

eg: A = [12, 3, 46, 34, 115]

⇒ Rng-Min(i, j) = 34

Made with Goodnotes

Applied in: Computational Geometry / string matching / other algorithms.

Ideas: (i) Sort, A in n.logn time

⇒ each query takes #step = 1

but, sorting may be not allowed

(ii) Rng-Min-brute(i, j):

Sequentially search for min in the interval.

↳ #steps = n

(iii) Rng-Min-Mat(i, j):

Store Rng-Min(i, j) in the (i, j)-th entry of a n×n matrix - M.

↳ M requires 1-time cost

↳ very big

▷ time & space $\approx n^2 \approx 10^{16} \times 1ns = 10^7 s$

Qn : is there a ds that solves Rng-min(i,j) in efficient time & space?

Q Find a super fast ~~algo~~ algo for $F(n) \bmod 2024?$

$2024 = m$

- IDEAS : (i) itFib(n, m) :→

for (i=2 to n)
  F[i] = F[i-1] + F[i-2] mod m;

▷ It Fib(n,m) requires $> \underbrace{3n}$ instructions & only $n \cdot \log m$ space.

(linear but bad)

(ii) RecFib(n,m): if (n≥2)

else return RecFib(n-1,m) + Rec(n-2,m)

▷ RecFib(n,m) requires > F(n) steps $> 1.5^{n-1}$

(exp time)

(iii) Clever insight: Do Double instead +1!

$$\begin{pmatrix} F(i) \\ F(i-1) \end{pmatrix} = \begin{pmatrix} F(i-1) \\ F(i-2) \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

evol^n of f as a 2x2 matrix.

$$A := \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\boxed{\begin{pmatrix} F(i) \\ F(i-1) \end{pmatrix} = A^{i-1} \cdot \begin{pmatrix} F(1) \\ F(0) \end{pmatrix}}$$

$\boxed{\triangleright\ F(n) = (A^{n-1})_{1,1}}$ Formula

$\triangleright\ F(n) = (A^{n-1} \bmod m)_{1,1}$

Insight of squaring: Square instead of multiply

$- \{ \quad A, A^2, A^4 \ldots\ldots\ A^k \bmod m$

[Repeated squaring]

$\triangleright$ Get to 2^k-power in k-steps.

$\triangleright$ Get to $(n-1)^{th}$-power of matrix in [logn] steps.

$\triangleright$ Computing $B \times C$ mat m $\quad (4^2+4)$ instructions.

$\Rightarrow 20 \log m$

▷ $A^{2^k} \bmod m$ takes $20k \log m$ steps.
($4 \log m$ space)

- **CleverFib(n, m):** $S[0] \leftarrow A := \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$;
$k \leftarrow \lceil \log(n-1) \rceil$,

· Write $(n-1)$ in binary $\rightarrow b_0 + b_1 2 + b_2 2^2 + \dots + b_{k-1} 2^{k-1}$
// n-1 in binary

- for ($i = 1$ to $k-1$)
    $S[i] \leftarrow S[i-1]^2 \bmod m$ // Repeated squaring

- $B \leftarrow S[0]^{b_0} \times S[1]^{b_1} \times \dots \times S[k-1]^{b_{k-1}}$
// $B = A^{b_0 + 2b_1 + \dots + 2^{k-1} b_{k-1}} = A^{n-1} \bmod m$

- return B; // $B_{1,1} = F(n) \bmod m$

Made with Goodnotes

▷ CleverFib(n, m) takes only $2\log m \cdot \log n$ $\times 2$

$= \boxed{4 \log m \log n}$ steps & $\boxed{\log m \cdot \log n}$ space

logarithmic scale << linear << expo...

(log n) (n) (2^n)

Time Complexity of an Algorithm

- Def^n: no. of instructions reqd. in the worst case, as a f^n of input-size (say, n).

11/1

- Detour into multiplication $a * b \bmod m$ ≈ log m bits

→ takes $(\log m)^2$ time

→ Advanced algos take ≈ $\boxed{\log m \text{ time.}}$

- Ex: Algos | Input-size | Complexity.

Algos	Input-size	Complexity
(i) itFib(n, m)	$(\log m + \log n)$ bits	$\approx n \log m$ [exp]
(ii) cleverFib(n, m)	"	[quad] $\approx 40 \log m . \log n$
(iii) sort $A[0, \dots, n]$	n words	$\approx \frac{n^2}{2}$ in $\log n$ [almost linear] i.sort, m-sort
(iv) Test sorting	"	$\approx n$

Issue in comparisons: $40 \log n > 3n$

for small n!

But, ▷ $40 \log n << 3n$, for all $n \geq n_0 := 128$

→ Don't look at small n's, but growing n.

→ You should compare the asymptotics of the

f^n in n ; $n \to \infty$

Eg: Algo A runs in time $t_A(n) := 5n^2+n+10$

" B " " " $t_B(n) = n^2 - 1$.

" C " " " $t_C(n) = 100n^{1.5} + 1000$

Qn which among B, C is a better improvement over A?

C

▷ $\lim_{n \to \infty} t_B/t_A \to 1/5$; $\lim_{n \to \infty} \frac{t_C}{t_A} \to 0$

[Caveat: Beware of astronomical constants in practice.]

- While comparing:

Obs 1: Ignore additive/multiplicative constants.

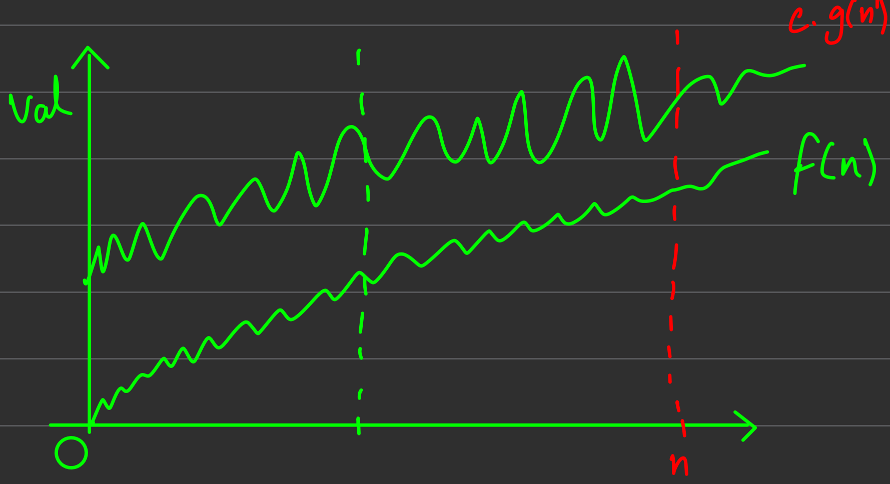
Obs 2: Identify the leading monomial

Order Notation

Defⁿ: Fns $f, g : \mathbb{N} \to \mathbb{R}_{>0}$

$n \to f(n), g(n)$

$f(n)$ is of the order of $g(n)$ if $\exists c, n_0 \in \mathbb{R}_{>0}$ $\forall n \geq n_0$, $f(n) \leq c \cdot g(n)$



$\Rightarrow$ write $f = O(g)$

Eg: $5n^2 + n + 10 = O(n^2)$ ← optimal

$100n^{1.5} + 1000 = O(n^{1.5})$ ← tight

$= O(n^2)$ ← loose.

$100 = O(1)$ ← Always in RHS

$[O(1) \neq 100]$!

<u>Proposition</u>: (i) $f = O(g)$, $g = O(h) \Rightarrow f = O(h)$

(ii) $f, g = O(h) \Rightarrow f+g = O(h)$

$f \cdot g = O(h^2)$

(iii) $f = O(h)$, $g = O(1) \Rightarrow f \cdot g = O(h)$

- Eg: $f = 3^n$, $g = 2^n \Rightarrow g = O(f)$; $f \neq O(g)$

$t_c = O(t_B)$ & $t_B \neq O(t_c)$

↳ t_c is smaller ordr

↳ C is better

Asymptotics:

- Insertion-sort : $O(n^2)$ time.
  Merge-sort : $O(n \log n)$ time.
  Binary-search : $O(\log n)$ time.
  CleverFib(n,m) : $O(\log m \log n)$ time.

→ Designing (asymptotically) fast algorithms is an ART!

15/1

Problem: Max-sum subarray

WAP: { Input: Array A storing n integers
output: Subarray $B = [A_i \ldots A_j]$ with max possible sum.

Ideas (i) Max-sum-brute(A):

for $i, j = 0, \ldots, n-1$. {
Compute sum $A[i, \ldots j]$;
track the maximum sum; }

Made with Goodnotes

▷ #steps $\approx \sum_{i,j=0}^{n-1} (j-i+1) \leq O(n^3)$

$\approx \frac{n^3}{12}$ or $n^3/24$

<u>Qn</u>: Can you do it in linear time $O(n)$?

- reduce the no. of indices from $2 \to 1$.?

$(i,j) \mapsto j$?

Reduce the search-space

<u>Def</u>^{n}: $s[j]$:= sum of the max-sum subarray ending at j.

- Ex: A: 3, -5, 3, 8, 2, -4 $\quad j=5$

3 -2 3 11 13 9 $\to$ 13

▷ If we have computed $S[0,\ldots,n-1]$, then we can output max-sum subarray of A in $O(n)$ time.

Pf: Output $\max(s[0,\ldots,n-1])$ = max-sum subarray of A. ($\because$ each j appears in s)

- we can also store i.

Qn: Compute $s[j]$ in $O(1)$ time; given the earlier values?

$\Rightarrow$ gives an $O(n)$ time algo!

▷ If $B = A[i,\ldots,j]$ achieves the optimum $s[j]$, then there are two options:

$B' \sqcup \{A[j]\}$ $\qquad\qquad$ $\{A[j]\}$ $\qquad$ X

▷ $B' := A[i,\ldots,j-1]$ achieves the sum $s[j-1]$

Pf:
- To get $s[j]$ either you use prefix before $A[j]$, or you don't
- The latter if done $\Leftrightarrow s[j-1] < 0$.

Lemma: (update) $s[j-1] > 0 \Rightarrow s[j] = s[j-1] + A[j]$

& $s[j-1] \leq 0 \Rightarrow s[j] = A[j]$

Pf: • In case-1 : $A[i, \ldots, j-1]$ extends to $A[i \ldots j]$

• In case-2 : " " is modified completely to $A[j]$.

Exercise: Use this to calculate $I[\,]$ array s.t. $I[j] = i$, with $s[j]$ = sum of $A[i, \ldots, j]$.

Max-sum-subarray $(A[0, \ldots, n-1])$: $s[0] \leftarrow A[0]$

$O(n)$ {

. for $j = 1, \ldots, n-1$.

if $(s[j-1] > 0)$ $s[j] \leftarrow s[j-1] + A[j]$;

else $s[j] \leftarrow A[j]$;

}

$O(n)$ $\begin{cases} \cdot \text{ Find } \max_j s[j] & \text{// say } s[j_0] \\ \cdot \text{ Output } (j_0, s[j_0]); & \end{cases}$

▷ # time $\simeq O(n)$.

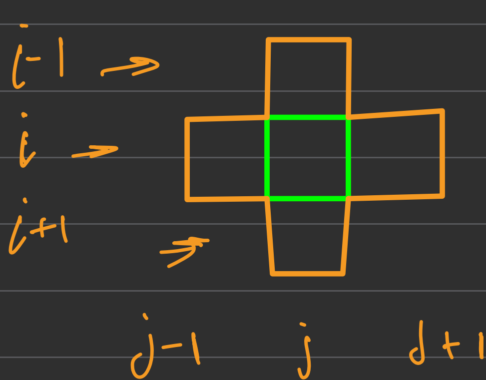
17/1

Problem: Local-Minima in a Grid

WAP: Input: $G[n][n]$ storing $n \times n$ distinct integers.

Output: Find a local-minima, i.e. an entry $G[i][j]$ smaller than its four neighbours

▷ A local minima exists



Pf: Because (global minima) exists.

↓

is also local minima.

Ideas??

(i) Single scan of G yields a local minima. Time $= O(n^2)$.

Q(W) Can you reduce it? $O(n)$?

(ii) Local exploration / iteration:

Go to a smaller neighbour and repeat.

localExplore(G[ ][ ]):

- c := G[0][0];
- while (c is NOT local-minima)

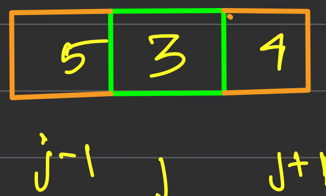
▷ In this algo, no cell is visited twice,

Pf: Value c decreases in every step ⇒ while-loop halts.

Exercise: $\exists$ input on which time $\approx O(n^2)$

(ii) let's make this idea more clever by looking at 1-D array $A[0, \ldots, n-1]$

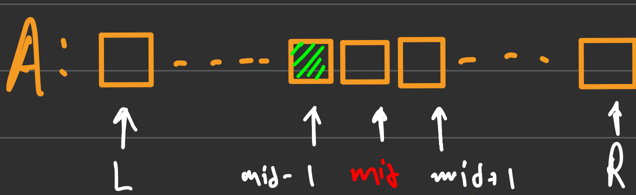
Ideas??



locally explore using $mid = \frac{L+R}{2}$

- If $A[mid]$ is not local minima in $A[L, \ldots, R]$, then move to a neighbouring smaller element:

$\boxed{mid-1 \text{ and } mid+1}$; move to that half.



say smallest of three

Left half is used next;

Lemma 1: $R \leftarrow mid - 1$

$\triangleright \; |L_{new} - R_{new}| \leq \frac{1}{2} |L - R|$

Pf: since we consider mid each time.

- The process ensures the following invariant.

Lemma 2: $A[L-1] > A[L]$; $A[R] < A[R+1]$

Pf:
- If $L-1$ or $R+1$ don't exist, then it is vacuosly true.
- When they exist, the pseudocode iterates. following these inequalities.
- So, we get a proof by induction on the # iterations.

Theorem: this algo finds local-minima $(A[0,...n-1])$ in $O(\log n)$ steps

Pf:
- By lemma, there are $\log n$-steps
- At last time $L = R$, which means that $A[L] = A[R] =$ single element (x).
- By lemma-2 the property is $A[L-1] > x < A[R+1] \Rightarrow$ 'x' is local minima.

Local-min-Array[0 .. n-1]:

- $L=0$, $R=n-1$, found = false

```
while (!found) {
        mid = (L+R)/2,

        if(A[mid] is local min) found = True.
        else if (A[mid-1] < A[mid]) R = mid-1
        else  L = mid + 1
}
return A[mid].
```

Q) Can we extend it to 2d grid?

- Instead of a cell in an array think of column in the grid.
- Consider $G[*][L-R]$; $mid = \left(\frac{L+R}{2}\right)$ the sub-matrix.
  ↳ rows
- Find min in $G[*][mid]$ ⇒ require $O(n)$-time.
  $= \min_{i=0}^{n-1} (G[i][mid])$

Say that 'min' is $G[r][mid]$

• Find $P = \min\{G[r][mid-1], G[r][mid], G[r][mid+1]\}$

if $P = G[r][mid]$
  output P.

Say $P = G[r][mid-1]$ then we take left half then change $R = mid-1$

- Formulate an invariant, similar to 1D lemma 2.

Lemma: $G[*][L-1] > G[min_L][L]$; $G[min_R][R] < G[*][R]$

This hold at every step of the recursion.

$min_L \to$ row where $a[*][L]$ resides.
$min_R \to$ " " $a[*][R]$ "

Pf:- Eg. Step as done above that updated R, by moving to the left-half.
- $a[min_R][R] < a[r][R] < a[r][R+1] < a[*][R+1]$.
  ($R+1$ ↑ mid old.)

⇒ 2nd bound. [1st is symmetric].
- This covers the induction step.

Ex:- Identify the base case.
⇒ The pf. is finished by the induction on number of columns.

Lemma (no. of steps):- This algo halfs the no. of columns and stops in $O(\log n)$ steps.
- $a[min_L][L]$ is a local minimum when $L = R$.

Pf:- $a[*][L-1] > a[mid_L][L] < a[*][R+1]$, and $a[mid_L - 1][L] >$ " $< a[mid_L + 1][L]$

⇒ $a[mid_L][L]$ is a local minima in the end $L = R$.

- the no. of recursive calls are $\log(n)$ as we are halfing the column each time.

Theorem: Local-minima in $a[n][n]$ is found in $O(n\log(n))$ time.

Pf: $O(\log(n))$ calls by prev. Lemma.
In each round it takes $O(n)$ time to find the min in mid-col.

$\Rightarrow \sum_{i=1}^{\log(n)} (n) \leq n\log(n)$.

- Further ideas: Reduce $\sum_{i \leq \log(n)} n = n\log(n)$ to $\sum_{i \leq \log(n)} n/2^i \leq O(n)$

- we can try this by taking the mid-row and picking upper/lower halves!

- Alternate b/w halfing no. of columns and number of rows.

- How to argue, what's the invariant?

Qn: Does the previous algo. directly works?

Exercise ÷ Find an input instance when the col-boundary invariant followed by row- " " doesn't give the desired results. (find counter example).

↳ col-invariant may get violated when we use middle-row and same for column.

Rough

TA-1

① (i) n

(ii) nlogn

(II) Array A , size n

```
while(L < R) {
        int mid = (L+R)/2 ;
        if (A[mid] = 20)  L = mid+1;
        else  R = mid;
}
return L;
```

(III) Using pointers in front of both arrays.