RANDOMIZED QUICKSORT

⇒ Avg. run time of Q.S → $\theta(n \log n)$.

⇒ Elements already sorted - reduced to $\theta(n^2)$

↓

One way to avoid this is to introduce a pre-processing step to randomize the I/p elements so that the alg. would always perform $\theta(n^2)$.

⇒. Choose the pivot element randomly.

```
randomizedquicksort A[first, last]
// I/p : Array A
// o/p : pivot element.
Begin
    if first < last then          → generate RN in this range.
        k = random (first, last)
        swap (A[first], k)
        pivot ← RSplit (A)        → select pivot randomly.
        randomized quicksort (A [first ... pivot])
        randomizedquicksort (A [pivot+1, last])
    endif
end.
```

complexity Analysis:

$A\{x_1 \ldots x_n\}$.

x_{ij} → random variable → whether two elements x_i and x_j are compared or not.

⇒ Question is to determine the numbers of counts, a given x_i, and x_j are compared.

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij}.$$

X → total no. of comparisons made by the algorithm

⇒ comparison of x_i and x_j would happen only under 2 conditions.

(1) A sub-problem of quick sort contains x_i and x_j

(2) Either x_i or x_j is chosen as pivot element.

Rewriting

$$E(X) = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[x_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr(x_i \text{ is compared } x_j)$$

⇒ Question of comparing x_i and x_j would arise only if either i chosen as a pivot element. Otherwise they would not be compared at all.

$Pr[x_i \text{ is compared to } x_j]$

$= Pr[x_i \text{ is chosen as a pivot}] + Pr[x_j \text{ is chosen as a pivot}]$

$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$

pivot is chosen from $j-i+1$.

$Pr[x_i \text{ is compared to } x_j] = \frac{1}{j-i+1} + \frac{1}{j-i+1}$

$= \frac{2}{j-i+1}$

probability $x_{ij} = 1$ is $\frac{2}{j-i+1}$

$$E(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[x_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$j - i = k$.

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k+1} \leq \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$= 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k} = \sum_{i=1}^{n-1} O(n \log n)$$

R.T - $O(n \log n)$

⇒ Complexity of random Q.S is similar to deterministic Q.S.

⇒ Randomized algs are very effective in solving problems.

⇒ fastness & effectiveness (Adv. of randomized alg.).

APPROXIMATION ALGS

Adv

⇒ optimize computer resources such as space and time.

⇒ Valuable tool for developing and evaluating different types of heuristics

⇒ Help to categorize problems based on their difficulty levels.

Disadv.

① focuses on worst case measures.

② Limited to only a certain set of problems or not applicable for decision problems.

Types ① Exact. ③ Heuristic.

② Approximate

STRING MATCHING ALGORITHM / PATTERN MATCHING.

⇒ In a given text T and pattern P, the pattern matching problem is to check whether P exists in T or not, and also where P appears in T

⇒ Text — length n.
⇒ pattern — length m

checks m characters from the 1st position of the text.

① If there is a mismatch, pattern is shifted by one position.

② If there is a match, then the next character is checked.

⇒ shift say k is the offset required to align the 1st character of the pattern with the k+1 character of text T.

⇒ shift k provides an exact matching is called a valid shift.

↳ Search process continues till the pattern is found or the text is fully exhausted

Text T = 'THIS'

pattern P = 'IS'

Index	1	2	3	4
Text	T	H	I	S

Index	1	2	3	4
Text	T	H	I	S

pattern P | I | S | (shift 0)

pattern P | I | S | (shift 1)

Index	1	2	3	4
Text	T	H	I	S

| I | S |

shift (2)

pattern P.

Steps

(1) Read T & P.

(2) Align P with T & move from left to right.

(3) Repeat following steps till the entire pattern is found or end of text T is encountered.

(1) compare 1st char of text T & pattern P. If match, move to compare 2nd char . . . . If all characters of P match with those of T, report success & exit else goto step (4)

(2) Move the pattern down the text by

one character and Increment the shift no. Goto Step 3.

(4) If p is not present in steps 3 & 4, report failure & end.

$O(nm)$.

RABIN - KARP ALG.

⇒ string matching alg,
⇒ Uses 'block of characters' for comparing pattern p with text T.
⇒ Developed by Michael O. Robin and Richard M. Karp in 1987.
⇒ key idea is to process text T to a sequence of integers. pattern p is also converted to an integer.
⇒ Then this alg. Compares sequence of numbers of text T and the fixed integer that represents the pattern p, to identify the presence of pattern p in text T.

STAGES

① Given ~~Text~~ text and pattern are converted into small numbers called 'fingerprints' of the strings.

② fingerprints of the text & pattern are compared.

[A better fingerprint would avoid most of the comparisons].

Stage 1

⇒ Text T & pattern P with lengths n and m.

⇒ Rabin karp alg. uses a fn. f to convert a block of characters of pattern P and T into numbers.

$f_i = f$ (substring (T, i, m)

m characters of the text starting from index i are taken as substrings and a fingerprint f_i is created.

Similarly pattern a converted to a fixed number to.

$P = p_1 p_2 \ldots p_m$.

fingerprint $f_0 = p_1 \times 10^{m-1} + p_2 \times 10^{m-2} + \ldots + p_{m-1} \times 10 + p_m$.

Evaluation of the polynomial

First m characters of text T = ($T = t_1 t_2 .. t_m$)

can be denoted similar to those in the pattern $f_1^t = t_1 \times 10^{m-1} + t_2 \times 10^{m-2} + .. t_{m-1} \times 10 + t_m$

① Reduced to a f_1^t

② Compare.

If t_i is the fingerprint of 1^{st} m characters of the text then new f_{i+1}^t for next m would be

$$f_{i+1}^t = (f_i^t - 10^{m-1} \times t_i) \times 10 + t_{i+m}$$

$f_i = f($ substring $(T, i, m))$ yields a larger value, a modulo arithmetic can be used to reduce the large no. to small no.

Eg. 9021 % 13 → 4. [small no. computation is considerably reduced]

Stage 2

⇒ Matching phase where the hash code of the text whose width is m are extracted and matched with fixed digit T.

4	1	3	3	1	2	4

Text

1	2	4

pattern.

(18)

pattern size - 3

$124 \bmod 13 \Rightarrow 7.$

Instead of 13, some other prime numbers can also be used.

Initially digit whose width is same as pattern is considered and modulo 13 operation is applied.

$413 \bmod 13 = 10.$

413 & 124 do not match.

Move to select 133 — new fingerprint f_{i+1}.

$T = 4133$ (i.e $t_1 = 4$ $t_2 = 1$ $t_3 = 3$ $t_4 = 3$)

Let $f_1^t = 413$

$f_{i+1}^t = (f_i^t - 10^{m-1} \times t_i) \times 10 + t_{i+m}$

$f_2^t = (f_1^t - 10^2 \times t_1) \times 10 + t_4$

$= (413 - 400) \times 10 + 3 = 133.$

SATISFIABILITY PROBLEM & COOK'S THEOREM

⇒ Satisfiability (SAT) problem is NP Complete.

⇒ SAT & NP-Complete was obtained due to the efforts of Lewin & Cook

⇒ Keep reduced of optimization problems such as * Hamiltonian tour
* Vertex Cover
* Clique
* SAT.

SAT

→ significant prob.

⇒ Given Boolean expression E having n variables $x_1, x_2, \ldots x_n$ & Boolean operators, is it possible to have assignment for variables true or false such that binary exp is true?

⇒ also known as formula - SAT / simply SAT.

[Takes Boolen exp and checks whether the given exp. is satisfiable]

Boolean variable — x have 2 value - T/F

Literal - Can be a logical variable say x or

negation of it, that is $\bar{x}$ or $\neg x$. (12)

$x \rightarrow$ positive literal

$\bar{x} \rightarrow$ negative "

clause : Seq. of variables $(x_1, x_2 \ldots x_n)$ can be seperated by a logical OR operator

Eg. $(x_1 \vee x_2 \vee x_3) \rightarrow$ clause of 3 literals.

Expression : preceding clauses are combined using boolean operator to form an expression.

CNF form.

⇒ An exp is in CNF (Conjunctive Normal form) if the set of clauses are seperated by an AND ($\wedge$) operator while the literal are connected by an OR ($\vee$) operator.

Eg. $f = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_3 \vee x_2)$.

3 CNF form.

Clause, 3 literals.