Can we sort - less than $(n \log_2 n)$? in the worst case?

$n \log n$ / $\log n$ /

If we are comparing (comparison) based algorithm - two elements

↓

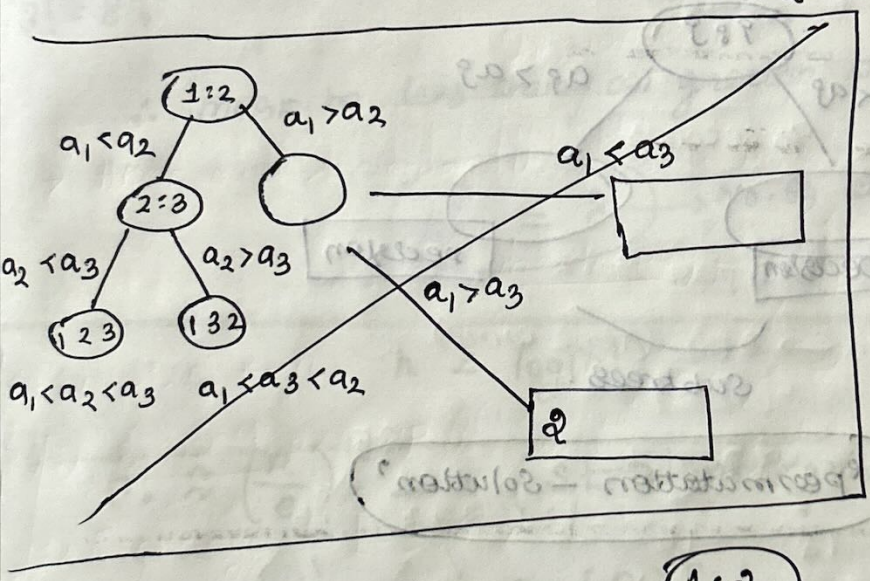
Then answer is No.

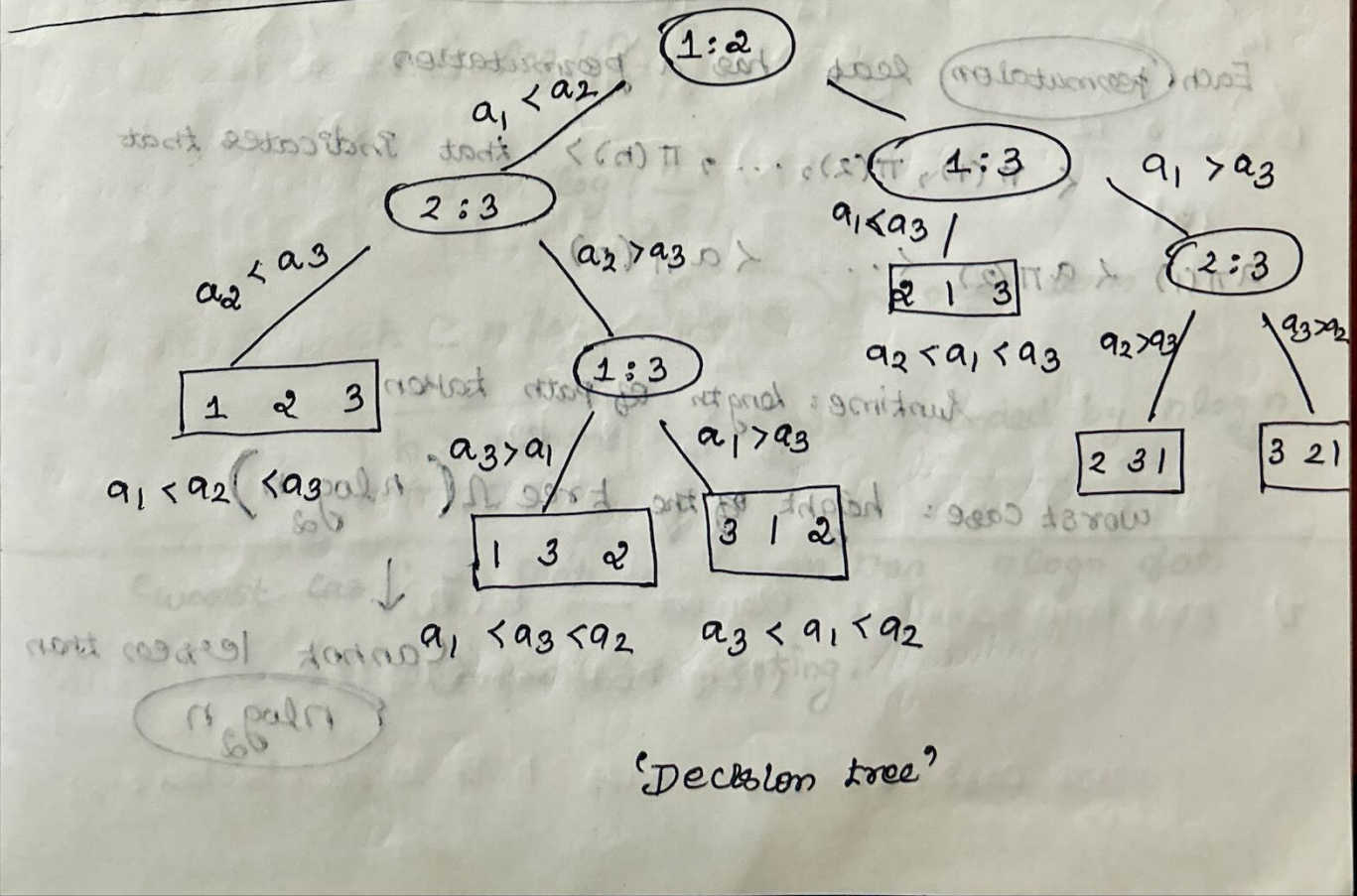
For comparison based : $\theta(n \log_2 n)$ → Best worst case scenario.

Decision tree.

$\langle a_1, a_2, a_3 \rangle$ → Sort → Tree (distinct numbers)

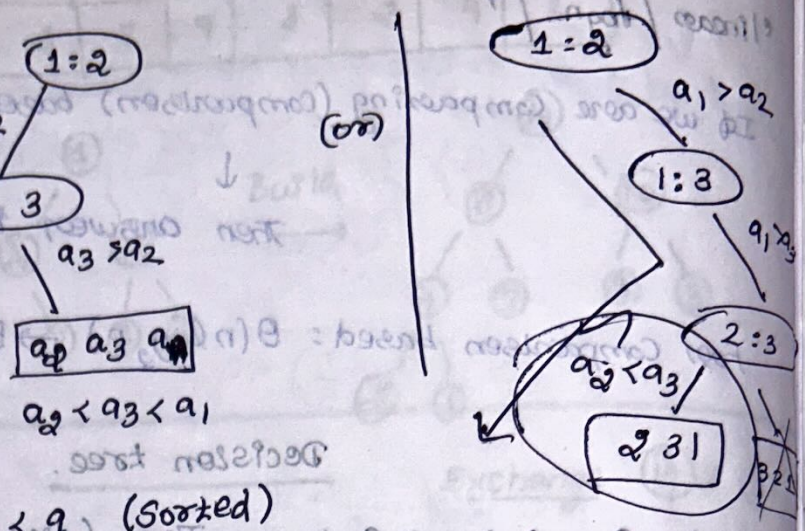
1:2 → (Distinct) → $1 > 2$, $2 > 1$) Two options.





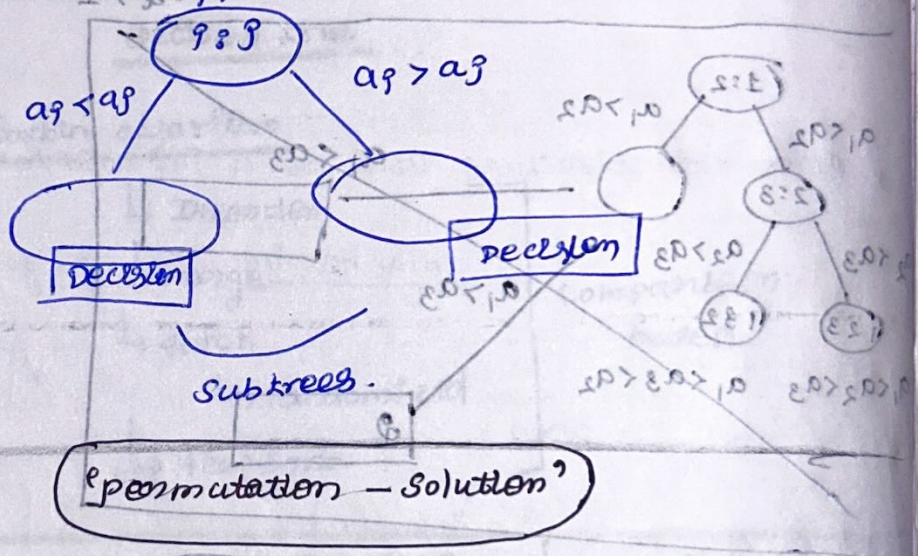
'Decision tree'

$a_1 = 9,\ a_2 = 4,\ a_3 = 6$



$4 < 6 < 9$ (Sorted)

Common



until it reach leaf node – every node has permutation'

Each permutation leaf has a permutation that indicates that

$\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$

$a_{\pi(1)} < a_{\pi(2)} < \ldots < a_{\pi(n)}$

Runtime: Length of path taken

worst case: height of the tree $\Omega(n \log_2 n)$

↓

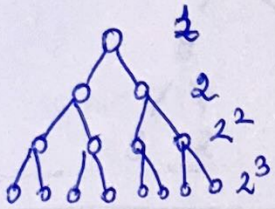
Cannot lessen than $n \log_2 n$

proof: Any decision tree that can sort n elements must have height $\Omega(n \log n)$

sol: # of leaves: (each leave: decision) = $n!$

Binary tree: Each node has two children.

h (height) → $\boxed{2^h \geq n!}$



→ Not a complete binary tree. Some may end sooner than others.

$3! = 8$,

∴ must be less than or greater than 2^h

$2^3 \leq 8$ → No. of leaves can be a max of 8.

Not more than 8.

$h \geq \log(n!)$

$n! \approx \left(\frac{n}{e}\right)^n$ → (Stirling's formula)

$h \geq \log\left(\frac{n}{e}\right)^n$

$h \geq n \log\left(\frac{n}{e}\right)$

$h \geq n \log n - n \log e$ low order term

$\boxed{h \geq n \log n}$ → height bounded by $n \log_2 n$.

"worst case" → can't be lower than $n \log n$ for Comparitive Sorting.

Assignment-3

1) worst case : when array is sorted / Reverse sorted order

2) Auxiliary space complexity of randomized quick sort : $O(\log n)$
(quicksort calls itself $\log n$ times)

3) 200 sec (min) ⟶ 1024 element sort (quicksort)

512 elements ⟶ ?

$$n \log n = 200$$

$512 \log 512 = 4608$

$1024 \log n = 200$

$n \log n = \frac{200}{1024} \; 1024 \log_2 1024$

$n \log_2 n = 10240$

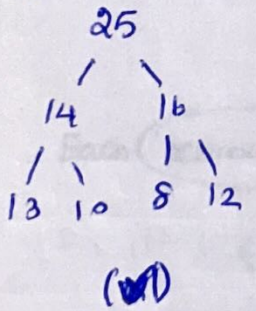
$$\frac{10240}{4608} = \frac{200}{x}$$

$$x = \frac{200 \times 4608}{10240} = 90 \text{ seconds}$$

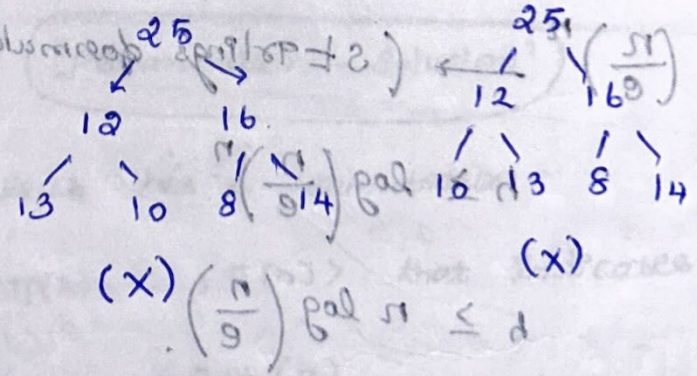
4) Randomized quicksort - Inplace sort.

5) Build heap operation ⟶ $\theta(n)$ (Time comp)

6)

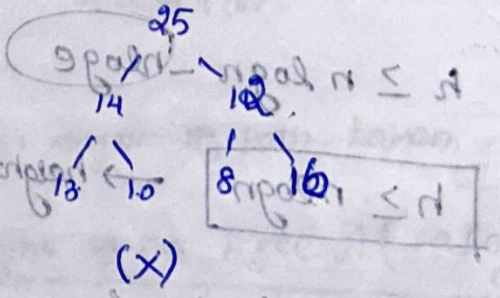


(✓)



(X)

(X)



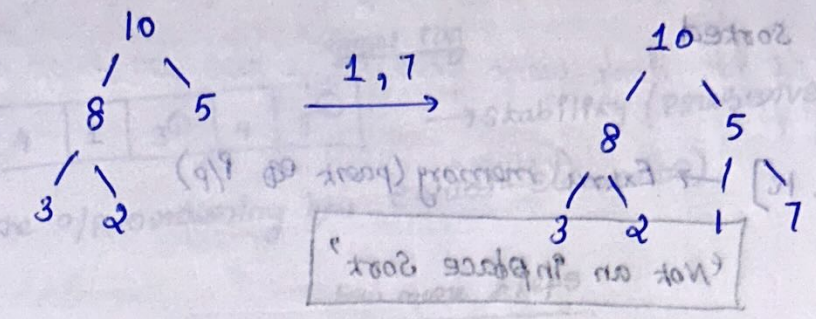
(X)

7) max heap Smallest element ⟶ $\theta(n)$

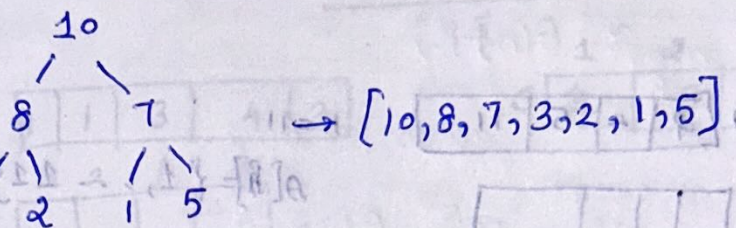
(leaf node - smallest)

worst case : all leaf nodes need to be checked.

8) 10 8 5 3 2



Not an in place sort



9) Any decision tree that can sort n elements $\geq \Omega(n \log n)$ in worst case alone.

10) Index of parent = $\lfloor i/2 \rfloor$

Linear time Sorting

Counting sort: (Not a Comparison based sorting)

$A[1, \ldots n] \rightarrow$ value of the number (bound)

range — fix the range

$A[1, 2, \ldots n]$ where $A[i] \in \{1, 2, \ldots k\}$

mention range!

* we need to take Auxillary array!

output = $B[1, \ldots n]$ Sorted

Auxillary storage : Bucket ... $C[1, \ldots, k]$

Based on value of K: we need to take this extra memory.

This storage is also the part of the o/p

$A[i] \in \{1, \ldots . k\}$

k → maximum number allowed to take (Range)

* Fixing the range - 1 criteria of this algorithm

O/P

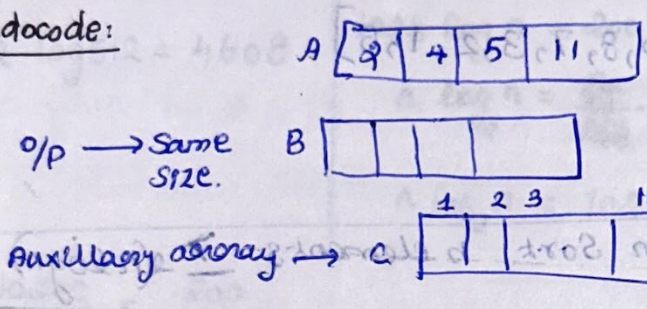
* B [1, .. n] sorted

Auxillary Storage

C [1, ... K] → Extra memory (part of I/P)

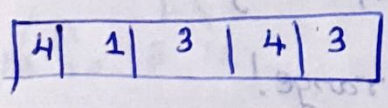
"Not an inplace sort"

Pseudocode:

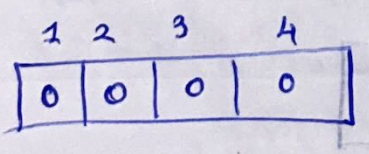


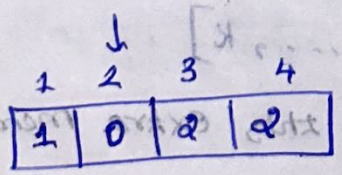
Counting Sort

1. For i ← 1 to K (Initialize C array)
2. C [i] ← 0
3. For j ← 1 to n — Count frequency of number.
4. do C[A[j]] += 1;
5. For i ← 2 to K
6. do C[i] ← C[i] + C[i-1]
7. For j ← n down to 1
8. do B[C[A[j]]] ← A[j]
9. C[A[j]] ← C[A[j]] - 1



maximum = 4





∴ 1, 3, 3, 4, 4.

(Sorted)

"we want some extra property" - Stability.

! Preserve the ordering of the equal element element.

1, 3, 3, 4, 4

we don't know which 3 comes first.

4	1	3	4	3

↳ This 3 must come first (Important for satellite data)

Small Tag

4	1	3①	4	3②

→ stability (preserve i/p ordering in the o/p ordering b/w equal elements).

Few more steps

eg:

A

4	1	3	4	3

C

1	2	3	4
1	0	2	2

B

1	2	3	4

'More Step' - Cumulative frequency

C

1	2	3	4
1	1	3	5

C[2] = C[2] + C[1]

= 0 + 1

= 1

upto index 2 → 1 element

upto index 3 → 3 elements

upto index 4 → 5 elements

Next Step

B

	3	3	4	4

B[C[A[5]]] ← A[5]

B[C[3]] ← A[5]

B[3] ← A[5]

B[3] = A[5]

B[3] = 3

C[3] − − = 1

C[3] = 2

1	1	2	5

2nd ← 3 goes 2nd.

B[C[A[4]]] ← A[4]

B[5] ← 4

C[4] − − = 1

1	1	2	4

B[C[A[3]]]

B[2] = A[3]

1②	3	3	4①	4

B[C[A[2]]] = A[2]

B[1] = A[2] = 1

C[1] = 0

0	0	2	4

'Sorted – as well as Stable? (Correct ordering)

* I/p order preserved in the O/p ordering.

'If we don't want Stability' – we can stop at step 4

Analysis

1, 2] → O(K)

3, 4] → θ(n) ≈ θ(n+K)

5, 6] → θ(K)

7, 8, 9] → θ(n)

when K = θ(n)

Time T(n) = θ(n+K)

= θ(n) → Linear

Linear → Range is also bounded by n

If K = n²

Time = θ(n+K) [K = n²]

Time = θ(K) = θ(n²)

when f(n) > n^(log_b a)

T(n) = θ(f(n))

'Asymptotic notation'

* when K = 5n, 100n, ... → This algorithm useful.

Radix sort – Herman (1890 – Invented)

(Used for US Sensor)

Idea: digit by digit sort!

original version: Most Significant digit to LSB

(Drawbacks)

↓ modified

Current: LSB to MSB.

Radix sort → digit by digit

→ LSB to MSB

'use Auxiliary Sorting algorithm' (Stable)

3	2 9
4	5 7
6	5 7
8	3 9
4	3 6
7	2 0
3	5 5

Stack with LSB →

USO stable sorting algorithm (Linear time)

7	2 0
3	5 5
4	3 6
4	5 7
6	5 7
3	2 9
8	3 9

(Stability)

→ Again use ...

7	2 0
3	2 9
4	3 6
8	3 9
3	5 5
4	5 7
6	5 7

→

3	2 9
3	5 5
4	3 6
4	5 7
6	5 7
7	2 0
8	3 9

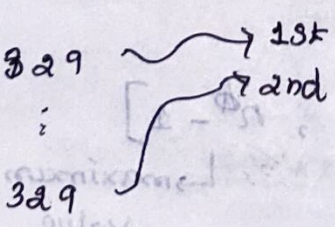
Sorted! Linear time → digit by digit sort.

Correctness (Induction method)

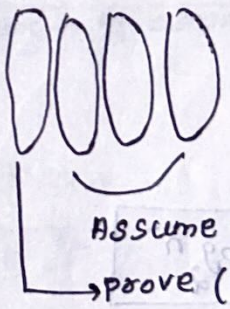
Correct upto $k-1$ → Assume

Correct upto k → Prove

"Different elements' – No issue



(First 329 must come first)



"Since – we are using stable sort algo (linear time)"

so it must be correct

"it also correct!"

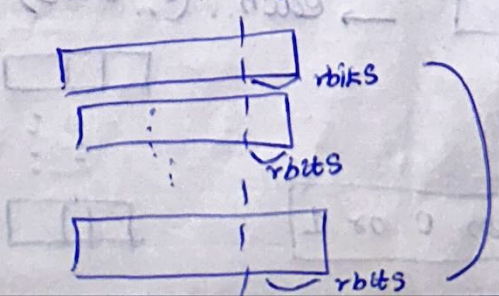
$45 \approx 2^6$

6 bits required

LSB: 0 to 9

max: 4 bits

Analyze (n elements – b bits) (binary)



$\#\,digits = \frac{b}{r}$ (b → Total bits in number, r → Single decimal digit)

r → Something we fix

Sort r bits at LSB

r bits at (LSB − 1) before LSB

Counting Sort

$\theta(n+K)$

(Here K will be linear we are analyzing single bit)

K − maximum range → maximum no. in array.

$\therefore LSB \rightarrow 2^r$

r bit (each 1) value

$\therefore \theta(n+2^r)$

Time Complexity of radix Sort = $\theta\left(\frac{b}{r}(n+2^r)\right)$ → size of the digit.

→ no. of elements

(No. of bits / No. of bits per LSB)

If $n = \log_2 n$

$2^r = n$

$[0, \ldots, n^d - 1]$ → maximum value.

$T(n) = \theta\left(\frac{bn}{\log_2 n}\right)$

$= \theta(dn)$ (digits)

$T(n) = \theta(n)$

$2^b = n^d$ (b: bits, d: digit)

r → each (r bits) Take LSB

→ Take

Let K gonna be either 0 or 1

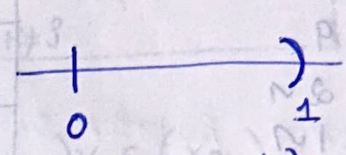
$\theta(n+2^r)$

$\therefore \theta(n)$

For $\frac{b}{r}$ digits. (or) d digits $\approx \theta(dn) \approx \theta(n)$

Bucket Sort

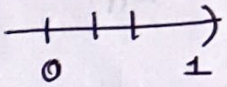
* Some numbers (Assume: numbers comes uniformly from the interval $[0, 1)$)



(uniformly)

Idea: (suppose n numbers)
$[0, 1)$
partition in to
n sub intervals.

each sub interval : Buckets



(n buckets)

Buckets:

$B[0], \ldots, B[n-1]$ → we put numbers in to the buckets

* ∴ we sort individual buckets.

Pseudocode

A → n-length array.

1) $n \leftarrow$ length (A)
2) For $i \leftarrow 1$ to
3) do Insert $A[i]$ in to $B[\lfloor n A[i] \rfloor]$
4) For $i \leftarrow 0$ to $n-1$
5) do Sort (Insertion) $B[i]$
6) $B[0] \,||\, B[1] \ldots \,||\, B[n-1]$ → Concatenate

n = 10, 0.78, 0.17, 0.26, 0.72, 0.89, 0.94,
0.21, 0.12, 0.23, 0.68

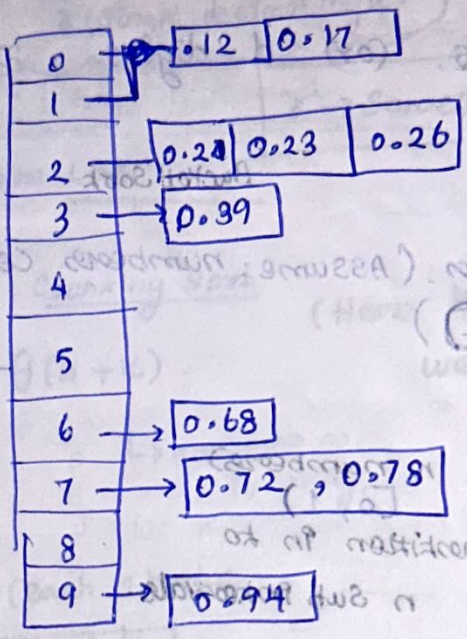
0.78
0.17
0.39
0.26
0.72
0.94
0.21
0.12
0.23
0.68

① Bucket

0
1
2
3
4
5
6
7
8
9

$\lfloor A[i] \times n \rfloor$ → round

0.78	7
0.17	1
0.39	3
0.26	2
0.72	7
0.94	9
0.21	2
0.12	1
0.23	2
0.68	6



First just fill the buckets, then sort the buckets individually using insertion sort. Then just concatenate.

Time complexity

$= \Theta(n)$ (Reading all elements - Fill the bucket)

$$+ \sum_{i=0}^{n-1} \Theta(n_i^2)$$

* $n_i \rightarrow$ number of elements in i^{th} bucket

$$E(T_n) = \Theta(n) + \sum_{0}^{n-1} \left(E(n_i^2) \right)$$

Expected value

$$E(T) = \Theta(n) + \sum_{i=0}^{n-1} O\left(E\left(n_i^2\right)\right)$$

Claim $E(n_i^2) = 2 - \frac{1}{n}$

Let $X_{ij} = \begin{cases} 1 & \text{if } A[j] \in B[i] \\ 0 & \text{otherwise.} \end{cases}$

$$n_i = \sum_{j=1}^{n} X_{ij} \quad , \quad (n_i)^2 = \left(\sum_{j=1}^{n} X_{ij} \right)^2$$

$$E(n_i^2) = E\left(\left(\sum_{j=1}^{n} X_{ij}\right)^2\right)$$

$$= E\left[X_{ij}^2 + \sum\sum_{k \neq j} X_{ij} X_{ik}\right]$$

$$\boxed{E(n_i^2) = E\left[X_{ij}^2\right] + \sum\sum_{k \neq j} E\left(X_{ij} X_{ik}\right)}$$

$$E(X_{ij}^2) = \frac{1}{n} + 0\left(1 - \frac{1}{n}\right)$$

$$\boxed{E(X_{ij}^2) = \frac{1}{n}}$$

$$E(X_{ij}) \times E(X_{ik}) = \frac{1}{n} \times \frac{1}{n}$$

$$= \frac{1}{n^2}$$

$$= n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} \rightarrow \sum\sum$$

$\therefore n \times (n-1)$

$$= 1 + \frac{n^2}{n^2} - \frac{1}{n}$$

$$= 2 - \frac{1}{n}$$

$$\therefore E(T) = \theta(n) + \theta(2) - \theta\left(\frac{1}{n}\right)$$

$$\boxed{E(T) = \theta(n)}$$

Order Statistics

$A[1, \ldots n]$ → Find ith smallest element.

$i = 1 \rightarrow$ minimum
$i = n \rightarrow$ maximum
$i = \frac{n}{2} \rightarrow$ median

Find ith smallest element.

Max (A)

1) Key ← A[1]
2) For $i \leftarrow 2$ to n
3) do If Key < A[i]
4) Key ← A[i]
5) Return (Key)

$\theta(n) \rightarrow$ Can't be extented to ith smallest/ largest.

* Input (A, i)
* output (ith smallest element)

Naive: Sort then return ith position.

Use comparison sort → we don't need to bound over i/p.

Linear Sorting - Restrict ourselves in a range of i/p.

1) Sort (A) → $\theta(n \log n)$ → quicksort
2) return (A[i]) → $\theta(1)$

Lazy - we can sort any smallest element

↓

No need for sorting.

Rank of an element.

A =	6	9	10	2	5	7

Rank(5) = position of that element in the sorted array.

2, 5, 6, 7, 9, 10

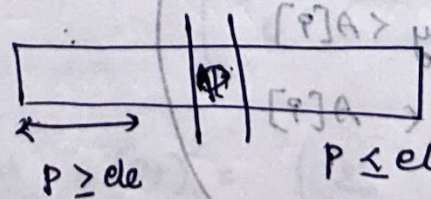
Rank(5) = 2

'Order Statistics problem' - Find ith smallest element.

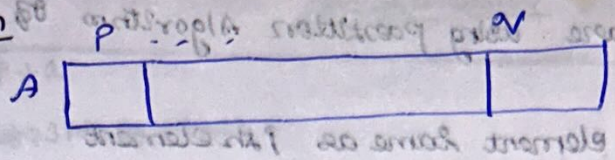
Find ith smallest = Find element with Rank = i

How to do it?

'partition' - Take that element as pivot



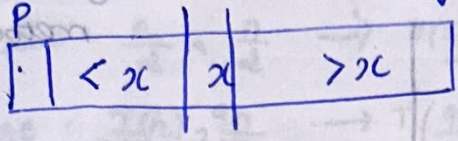
Partition



* $x \leftarrow A[p]$

$k = r - p + 1$

k = rank of x



∴ After step 1 → pivot will be r

∴ $k = r - p + 1$.

(If $p = 1$)

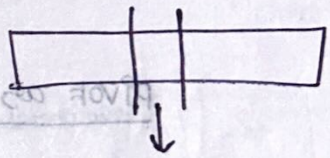
$k = r$ → Rank of x

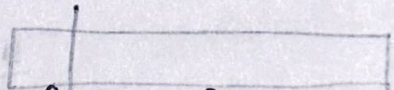
(correct position of x)

If $k = 7$ (7 th smallest element)

$q = 7$

∴ $q = 4$ → must be left to 7 th smallest element.





$q > 4$ → Right side of 7.

Select $(A[p, \ldots, q], q)$

① If $p = q$ return $A[p]$

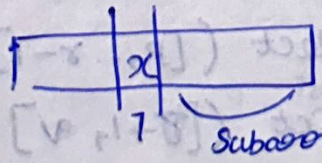
② $r \leftarrow$ partition $(A[p, \ldots q])$

③ $k \leftarrow r - p + 1$ // k = rank of x

④ If $(q == k)$ return (x)

⑤ If $(q < k)$ then Select $(A[p, r-1], q)$

⑥ If $(q > k)$ then Select $(A[r+1, q], q-k)$



Subarray (q^{th} means 2nd smallest in that subarray)

↓

$q - k$ (new one)

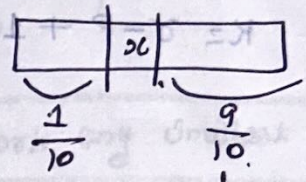
We are using partition algorithm of quicksort

* Lucky → partition element same as i^{th} element
* Unluck → Not

$$T(n) = T\left(\frac{n}{2}\right) + \theta(n) = \theta(n) \text{ by master method.}$$

Recurrence → splitting.

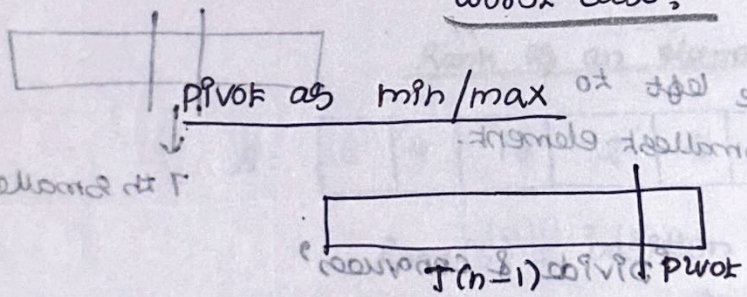
$T(n)$ = Not lucky



↳ Here falls

$$T(n) = T\left(\frac{9n}{10}\right) + \theta(n) = \theta(n)$$

worst case!



$$T(n) = T(n-1) + \theta(n)$$

$= \theta(n^2)$ ⟹ worst case!

Idea: Choose the pivot randomly!

Randomized Select (A, p, r) $[p \dots r]$

$\theta(1)$ ← 1) If $p == r$ return $A[p]$

$\theta(n)$ ← 2) $q \leftarrow$ Rand partition $(A[p \dots r])$

$\theta(1)$ (3) $k \leftarrow q - p + 1$ // k = rank(x)

4) If $(i == k)$ return (k)

Nature of Partition (5) If $(i < k)$ Rand Select $([p \dots q-1], i)$

6) If $(i > k)$ Rand Select $([q+1, r], i-k)$

Order statistics

* $k = r - p + 1$

$q < k$ (left side of pivot)

$q > k \rightarrow$ new $(p-k)^{th}$ element (partitioned another)

Worst case: $T(n) = T(n-1) + \theta(n) = \theta(n^2)$

Best case: $\frac{n}{2}, \frac{n}{2} \rightarrow T(\frac{n}{2}) + \theta(n) = T(n) = \theta(n)$ } partition.

Avg case: $\frac{1}{10}(n), \frac{9n}{10} \rightarrow T(\frac{9n}{10}) + \theta(n) = \theta(n)$

Randomized one

Rand_Select(A[p, ..., r], i)

good pivot: $\frac{n}{2}$ / Same portion $\frac{9n}{10} : \frac{n}{10} \rightarrow \theta(n)$

Worst case: $\theta(n^2)$

Average case analysis

$T(n)$ = Runtime of Rand-Select with n elements

↳ Random variable

0 to $n-1$ $\rightarrow T(n) = T(n-1) + \theta(n)$

1 : $n-2$ $\rightarrow T(n) = T(n-2) + \theta(n)$

k : $n-k-1$ $\rightarrow T(n) = T(\text{maximum of } \{k, n-k-1\})$

maximum of these two.

$$T(n) = \begin{cases} T(\max\{0, n-1\}) + \theta(n) & \text{if } 0 : n-1 \text{ split} \\ T(\max\{1, n-2\}) + \theta(n) & \text{if } 0 : n-2 \\ \vdots & \\ T(\max\{k, n-k-1\}) + \theta(n) & \text{if } k : n-k-1 \\ T(\max\{n-1, 0\}) + \theta(n) & \text{if } n-1 : 0 \end{cases}$$

∴ we don't known which partition will occur!

Every pivot is randomly Chosen?!

Take expectation!

Indicator Random Variable → For algebraic expression

$$X_K = \begin{cases} 1 & \text{if } K = n-K-1 \\ 0 & \text{otherwise} \end{cases}$$

If $0 : n-1$ is the partition $X_0 = 1$ All others 0

$X_1, X_2, \ldots = 0$

$P(X_K = 1) = \frac{1}{n}$ (equally likely)

$$T(n) = \sum X_K \{ T(\max\{K, n-K-1\}) + \theta(n) \}$$

$$E(T(n)) = E\left(\sum X_K \{ T(\max\{K, n-K-1\}) \} + \theta(n) \right)$$

$$= \sum \left(E(X_K \cdot T(\max\{K, n-K-1\}) \} + \theta(n) \right)$$

$E(WY) = E(W)\,E(Y)$ Independent [we are choosing randomly]

$$\sum \left(E(X_K)\, E\left(T\left(\{\max\{K, n-K-1\}\right)\right) + \theta(n) \right)$$

$P(X_K = 0) = 1 - \frac{1}{n}$

$P(X_K = 1) = \frac{1}{n}$

$$= \sum_{K=0}^{n-1} \left(\frac{1}{n} \right) E\left(T\{\max\{K, n-K-1\}\right) + \theta(n)$$

$$1 \times \frac{1}{n} + 0\left(1 - \frac{1}{n}\right)$$

$$= \frac{2}{n} \sum_{K=\frac{n}{2}}^{n-1} E\left(T(K)\right) + \theta(n)$$

$E(X_K) = \frac{1}{n}$

$K = 0, 1, \ldots, \frac{n}{2}, \frac{n}{2}+1$

Proof:

$E(T(n)) = O(n)$ $(E(T(n) \le c_n) \longrightarrow$ Take $c > 0$

Induction hypothesis

$$E(T(K)) \le c_K \quad \forall\ K < n$$

$$E(T(n)) = \frac{2}{n} \sum_{K=\frac{n}{2}}^{n-1} E(T(K)) + \theta(n)$$

$$\leq \frac{2c}{n} \sum_{K=\frac{n}{2}}^{n-1} K + \theta(n) \quad \text{(using induction hypotheses)}$$

Fact

$$\sum_{K=\frac{n}{2}}^{n-1} K \leq \frac{3}{8} n^2$$

$$\leq \frac{2c}{n} \left(\frac{3}{8}\right) n^2 + \theta(n)$$

$$\leq \frac{3c}{8} n + \theta(n)$$

$$\leq \frac{3}{4} cn + \theta(n)$$

choose c in such a way.

$\leq$ (less on)

$$= cn - \left(\frac{cn}{4} - \theta(n)\right) \rightarrow \text{positive}$$

$$\leq cn$$

$E(T(n)) \approx \theta(n)$ → Avg. Case

Worst case = $O(n^2)$

---

worst case linear time order statistics

Guarantee: the complexity for finding ith smallest element worst case

Invented by: Blum, Pratt, Rivest, Tarjan – 1973

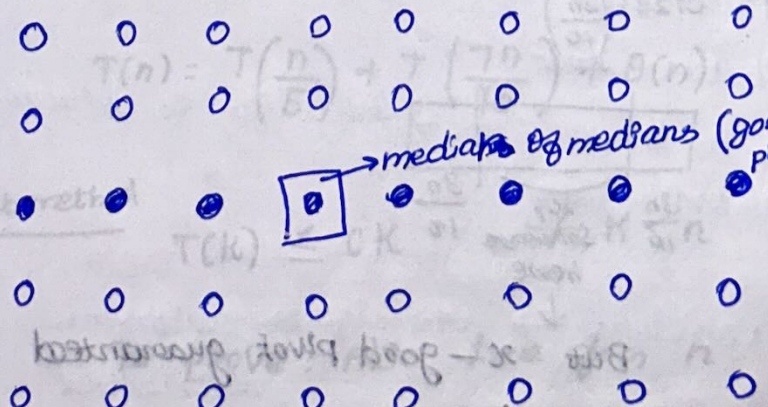
Idea: Generate a good pivot. (min/maximum) → bad pivot.

Good: Ensure some portion

'Avoid 0 : n-1' case!

How to make sure! Good pivot.

'Divide in to 5 element groups'

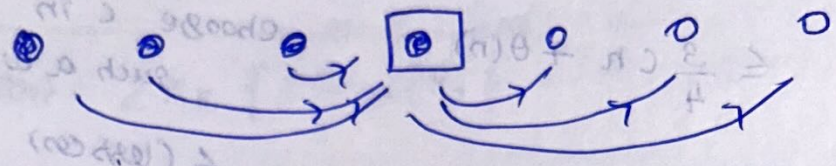


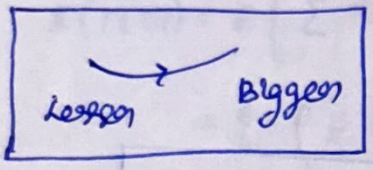
"find median"

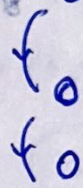
* Total number of groups : $\frac{n}{5}$ medians

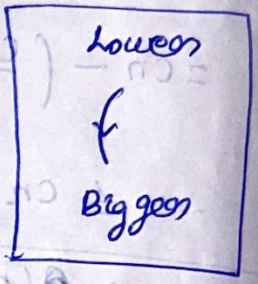
* medians of medians : pivot

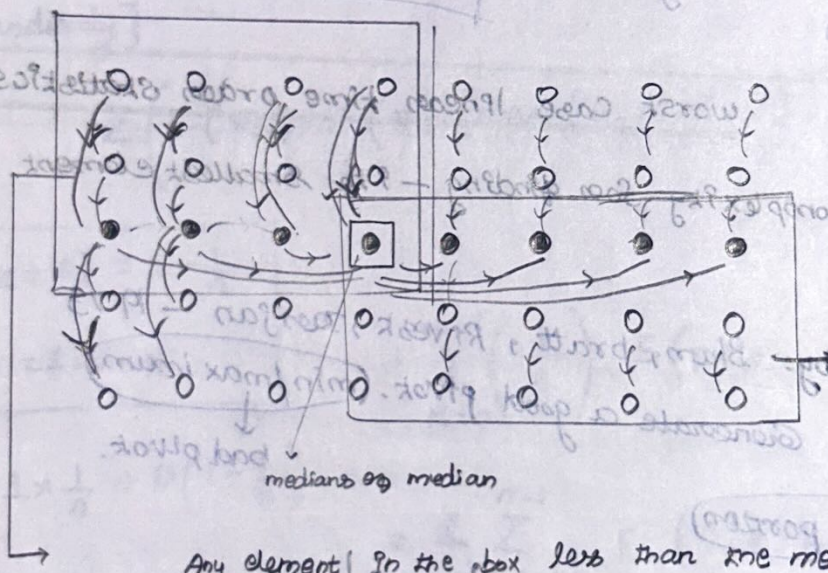
"what's the guarantee"







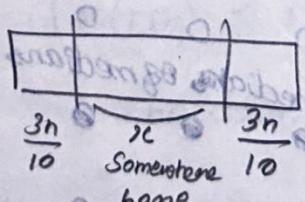




Any element in the box less than the median → Guarantee.
(equal to)

2 portions → ≤ x size : $\left(\frac{n}{10}\right) \times 3 = \left(\frac{3n}{10}\right)$

→ ≥ x size : $\left(\frac{3n}{10}\right)$



But x – good pivot guaranteed

∴ some position left side ≤ x
right side ≥ x

"Taking this good pivot" – calling partition" : Some position

not → 0 : n-1

"no – worst case" : $\Theta(n^2)$

SELECT(A[1,....n], i) Pseudocode SELECT(A[1,...,n], i)

1) Divide n elements into groups of 5. Find median of each group

2) Recursively select the median x of the $\lfloor \frac{n}{5} \rfloor$ group of medians. to be the pivot.

3) partition around the pivot x and then k be Rank(x)

(k = r − p + 1) ← rank k(x)

4) If (i = k) return x

5) If (i < k) SELECT (A[p, ... r−1], i)

6) If (i > k) SELECT (A[r+1, ... q], i − k)

① $\Theta(n)$ $\left[\frac{n}{5} \text{ groups}\right]$

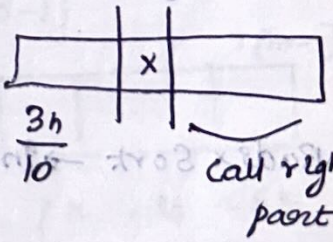
② $T\left(\frac{n}{5}\right)$

③ $\Theta(n)$ → partition

④ $\Theta(1)$

⑤ $T\left(\frac{7n}{10}\right)$ → maximum (Take)-right side

⑥



$T(n) = O(n)$

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \Theta(n)$$

Sub method

$T(k) \leq ck \quad , \forall k < n$

To prove : Also true for n

$$\theta(n) \leq \frac{cn}{5} + \frac{7cn}{10} + \theta(n)$$

$$= cn \leq \frac{cn\,(2+7)}{10} + \theta(n)$$

$$cn \leq \frac{9cn}{10} + \theta(n)$$

$$\leq cn - \left(\frac{9cn}{10} + \theta n\right)$$ → must be +ve

Choose c & n in such a way, (+ve)

$T(n) \leq cn$

$T(n) = O(n)$ → worst case runtime for our Select algorithm

'This case: pivot - Not randomized'

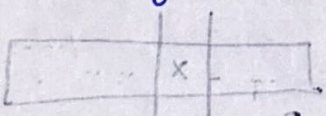
Assignments

1) $\{9, 7, 13, 28, 22\}$ → How many comparisons

↳ No comparison (0).

2) Auxillary space req eg counting sort → $O(n+k)$

k → range of i/p



3) Radix Sort → in place sorting? False (Bucket: extra space)

4) Bucket Sort: Not an in place sort

↳ considered as comparison based sort'

↳ distribution sort

↳ Runs in linear time - when i/p is drawn from a uniform distribution.

5) Buck sort - most efficient → when i/p is uniformly distributed

6) Runtime of SELECT algorithm is $T(n) = O(n)$ if groups = 5 (our case)

7) most efficient - to sort string (ASCII characters) → counting sort.

8) Lowest worst case time complexity of finding k^{th} smallest element of an unsorted array → $O(n)$ → using (good pivot tech)

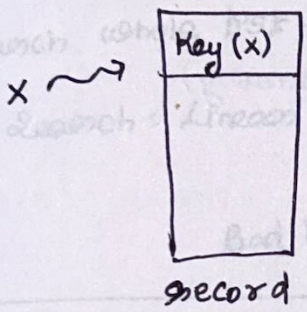
9) Radix Sort : Non-comparison based integer sort.

10) For numbers in the range 0 to $n^d - 1$, radix sort runs in $O(dn)$ time.

---

week-5 - Hash function - symbol table problem

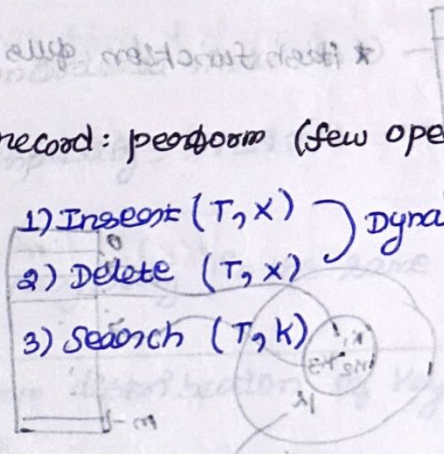
*Symbol table : T holding n records



maintain record: perform (few operations)

1) Insert (T, x)
2) Delete (T, x)
3) Search (T, k)

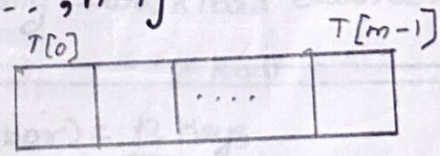
} Dynamic



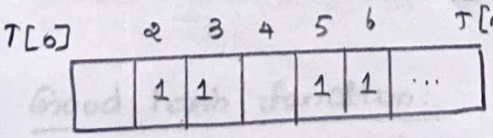
Heap (dynamic)
(discussed set)
(maximum)/min

Direct access table

Keys: coming from certain set $K \in \{0, 1, \ldots, n-1\}$

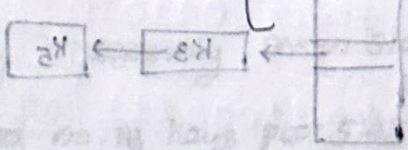


Array



$\{2, 2, 3, 6\}$

$T(k) = \begin{cases} x, & \text{if } x \in K \\ & \text{Key}(x) = k \end{cases}$



$I(k) = \begin{cases} 1 & \text{if } x \in K \text{ and } \text{Key}(x) = k \\ 0 & \text{otherwise} \end{cases}$

Time Complexity: Insert (T, 10) → $\theta(1)$

Delete (T, 5) → $\theta(1)$

Search (T, k) → $\theta(1) = \theta(1)$

Found (Go to that place - get)

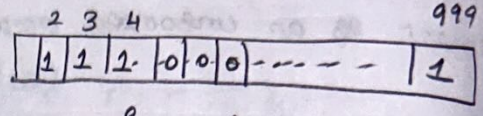
"Constant time operations"

Problem: Memory

$\{2, 3, 999, 4\}$

↳ 999 → structured size

(only 4 records)

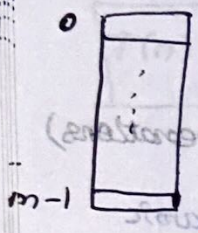


"worst"

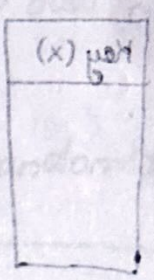
Hash function

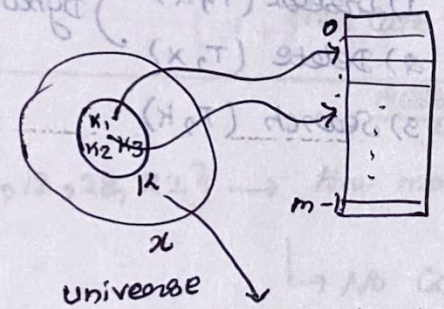
$h : u \rightarrow \{0, \ldots, m-1\}$

$u \rightarrow$ set of (universe) of key. — slots (table size available to store)



* Hash function fills using hashing





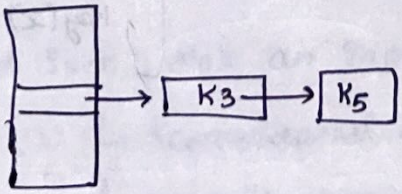
$h(k_3) = p$

$h(k_4) = s$

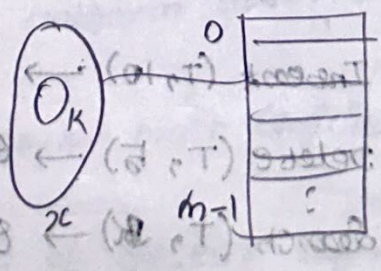
Collision: Same key can't (answer from two numbers same using hash functions)

Handle Collision:

* Chaining / Linked list.

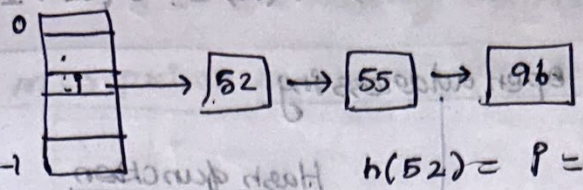


Analyze chaining method.



$m \rightarrow$ # slots

$k \rightarrow$ # key.

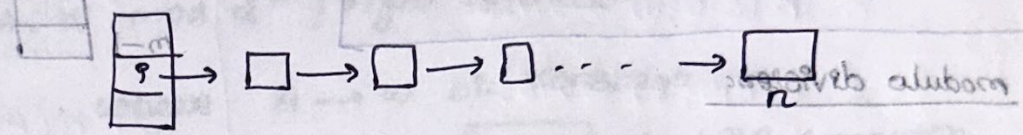


$h(52) = 9 = h(55) = h(96)$ [Say]

Search a key:

we found its in 9 → 9th has some numbers.

Say: n keys → all are in 9th slot



Search whole list → Not sorted (unordered linked list) → more time.

★ Search: Linear time (Time complexity = linear)

Bad hash function: Everybody in the same slot.

Good hash function: uniform distribution of keys over the slots. (n keys over m slots).

$$\#\text{keys per slot} = \frac{n \text{ keys}}{m \text{ slots}} = \frac{100}{10} = 10 \text{ (number of keys per slot)}$$

'Distribute uniformly' — workload balance.

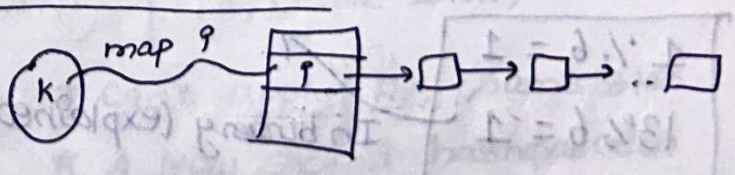
$\alpha = \#\text{keys per slot}$ (Load factor) $= \frac{n \text{ keys}}{m \text{ slots}}$.

Good hash function:

★ Distribute the key uniformly over slots

★ $\alpha = \frac{n}{m}$ = Expected no. of keys per slot

★ Search time: Search (K)



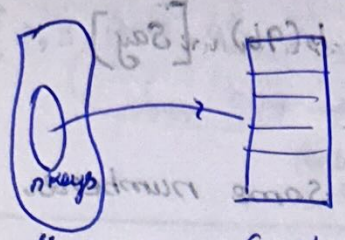
$\theta(1) + \theta(\alpha)$

chain size = α

$= \theta(1+\alpha)$ [Good hash function → Evenly (uniformly) distributed.

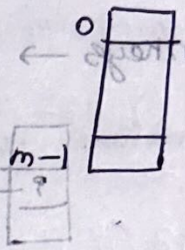
If $\alpha = 1 \rightarrow T(n) = \theta(1+1) \approx \theta(1)$

open addressing



Universe (Uniformly distributed)

$\alpha = \frac{n}{m}$



Hash function

I (Division method)

Assume: keys are integers

modulo division:

$h(K) = K \bmod m$

$m = 2^r$ Problem: $K = 1011000111\ \boxed{011010}$ (Binary) — $h(K)$

$m = 2^6$

$r = 6$

$h(K) = K \bmod m$

$K = k_l \ldots k_1 k_0$

$K = k_0 + 2k_1 + 2^2 k_2 + \ldots + 2^l k_l$

$= k_0 + 2k_1 + 2^2 k_2 + \ldots + 2^{r-1} + 2^r [k_r + \ldots]$

$K \bmod 2^r$ → $\boxed{011010}$ → This alone will come

1011000111 → It just don't cares!

while fixing (011010)

$1011000111\ \boxed{011010}$

all are colloiding into the same 9.

$1 \% 6 = 1$

$13 \% 6 = 1$

$19 \% 6 = 1$

In binary (explained)

[Huge Collision]

This is not a good hash function!

∴ So we usually choose

m → prime number which is close to $2^r / 10^r$

↓ No same modulo answer.

II) multiplication method

$m = 2^r$, number word ⇒ w bit words.

Key: w bits

$h(K) = (A \cdot K \mod 2^w)$ right shift $(w - r)$

where A → w bit constant

A = 64 → 64 bits computer.

r = 3

$m = 8$ slots $= 2^3 = 2^r$, $W = 7$ bits, $A = 1011001$

K = 1101011

$A \times K$ = 14 bits = 10010 10011 1001

mod 2^w

(1001 10011) shift $(w = 7 - r = 3)$

10 1 100 (1) right shift (4 times) = 011

$h(K) = 011$

Handle – collision – open addressing

* Strategy to handle collision → open addressing

*In chaining: linked list: Extra storage

* Suppose: no extra storage! (Restriction).

Open addressing!

* Hitting a occupied slot → Search for next slot. (Free slot)

* Can't apply the same slot

* A New second hashfunction (updated)

Second also collided → New one (third one).

$h: U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}$

Probe sequence:

$0: h(K, 0)$

Search a key!

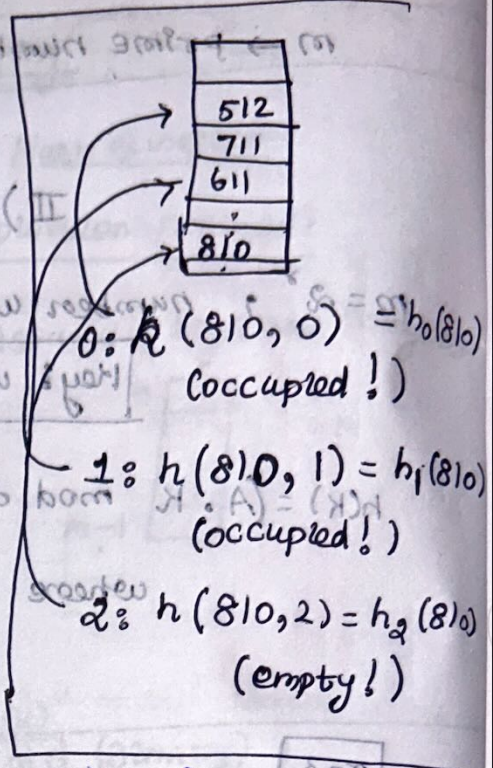
Search (810)

Follow same probe sequence

$0: h(810, 0) \rightarrow$ No!

$1: h(810, 1) \rightarrow$ No!

$2: h(810, 2) \rightarrow$ yes!

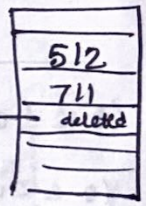


Idea: Search for empty one

Delete! – Tricky

empty slot.

Suppose we delete 611



Searching 810: when we hit empty (stopped!)

"we need to move on!" → Not originally empty – somebody deleted it"

"use some bit" – As identifier

put a tag! → "This room was not org. empty"

↓

Open addressing (sequence of probes)

Linear probing

$q: h(K, q) = (h(K) + q) \bmod m$

↓

qth probe

"see next one is available or not"

If yes – Just fill.

$0 : h(K,0) = h(K)$

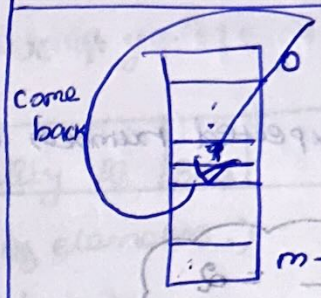
$1 : h(K,1) = h(K) + 1$

Drawback: clustering (not uniformly destributed)! → Solution

$\therefore (h(K)+1) \bmod m$

$= (h(K) + 1)$

'come back'



Double probing

2 hash functions

$q : h(K,q) = (h_1(K) + q\,h_2(K)) \bmod m$

$0 : h(K,0) = h_1(K)$

$1 : h(K,1) = (h_1(K) + h_2(K)) \bmod m$

$2 : h(K,2) = (h_1(K) + 2(h_2(K))) \bmod m$

If h_1, h_2 → Good hash functions → result good!

universal hashing

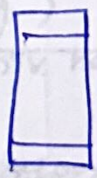
Analysis of open addressing: — By a theorem

★ Theorem: Given an open addressed hash table with load factor $\alpha = \frac{n}{m} < 1$ (no. of keys < number of slots) $n < m$ (No extra space)

★ the expected no. of probes in an unsuccessful search, is atmost $\left(\frac{1}{1-\alpha}\right)$.

Proof:

Probability: hitting an occupied slot $= \frac{n}{m}$



'n keys', 'm slots'

n keys.

'classical definition'

$$\#\text{ of probes} = 1 + \frac{n}{m}\left(1 + \frac{n-1}{m-1}\left(1 + \frac{n-2}{m-2}(1+\ldots)\right)\right)$$

Expected # of Probes $= \leq 1 + \frac{n}{m}\left(1 + \frac{n}{m}\left(1 + \frac{n}{m}\left(1 + \cdots\right)\right)\right)$

$\leq 1 + \alpha\left(1 + \alpha\left(1 + \alpha\left(1 + \alpha \cdots\right)\right)\right)$

$\leq 1 + \alpha + \alpha^2 + \alpha^3 + \cdots$

$= \frac{1}{1-\alpha}$

∴ Expected number of probes for unsuccessful search.

Now table is half filled $\alpha = \frac{1}{2} = 50\%$

$\frac{1}{1-\frac{1}{2}} = 2$ → 2 probes

$\frac{1}{1-\frac{9}{10}} =$ 10 probes.

'90% → 90 slots – 90 keys filled' – Almost loaded

'Good' – In handling collision.

There has to be collision → doming size is big than codoming size

---

Universal hashing

weakness of hash function:

'Always possible to have collision' → more time → try more keys (for collision)

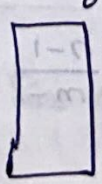
'Fundamental weakness'

Avoid this weakness!

* Quick-Sort – Know position (Build bad array)
* Random quick sort pivot – Impossible // Such thing.

Idea behind universal hashing!

Random choice of Hash function from a collection of hash functions.



H(Collection of hash functions)