A map based data structure that supports
1. search
2. insertion

For any given $n$, we represent the $n$ elements using a set of arrays $A_i$ where $|A_i| = 2^i$ such that $\sum |A_i| = n$.

For eg. $n = 11$ then we have $A_3, A_1, A_0$.

The elements in $A_i$ are sorted (but have no relation with $A_j, j \neq i$).

Given any $n$, we can determine $A_i$'s from the binary rep of $n$, similar to Binary Heap.

$n = 12$ $A_3, A_2$
Search: for a given key \( x \), we will do binary searches in each of the arrays.

Time: in \( A_i \) we take: \( O(i) \).

Max. time: \( \sum_{i=1}^{n} i \leq O(\log n) \).

Insertion: \( n \rightarrow n+1 \)

\[ A_{i_1}, A_{i_2}, \ldots, A_{i_k} \quad A_{j_1}, A_{j_2}, \ldots, A_{j_k} \]

Time to merge two sorted arrays \( S_1, S_2 \) with

\[ |S_1| = n_1, \quad |S_2| = n_2 \]

\[ = O(n_1 + n_2) \leq c \cdot (n_1 + n_2) \]

\[ : O(16) \]
In general, if we have done this over \( j \) stages, then cost will be \( O(2^j) \).

"Worst case": \( n = 2^j \)

Call arrays were replaced.

Total cost is \( O(n) \).

How often will this happen?

How often will \( A_i \) be rebuilt?

Once every \( 2^i \) insertions.

Cost of rebuilding \( A_i \) : \( O(2^i) \).

For inserting \( n \) elements, \( \frac{n}{2^i} \) rebuilds:

\[
\text{Total cost of inserting } \frac{n}{2^i} \text{ elements in all the arrays } \\
\leq \sum_{i=0}^{\log n} \frac{n}{2^i} \cdot O(2^i) \\
\leq O(n \cdot \log n)
\]

Average cost : \( O(\log n) \).
Stacks:

0. Insertion, push $O(1)$

1. Deletion, pop $O(1)$

2. Empty stack $O(\#\text{element})$
   pops all elements

Consider a sequence of push, pops and Empty Stack operations: $n \#\text{them}$

What is the total cost?

$\Rightarrow O(n^2)$

Consider a function

$\phi: \mathcal{D} \rightarrow \mathbb{Z}$

potential function

data structure

Amortised cost of an operation $O$, relevant to $D = \text{actual cost} +$

change in potential

Eg: Suppose for the stack, we define

$\phi(S) = (\#\text{elements in } S)^2$

A mostised cost of pop: $1 \cdot (\text{actual cost}) + 10^2 - 11^2$

if $S$ had 10 elements
Total Amortized cost of a sequence of operations $O_1, O_2, O_3, \ldots, O_n$

$$\sum_{i} \left( T(O_i) + [\phi_{i+1} - \phi_i] \right)$$

actual cost

potential change in potential after $i$-th op.

$$\leq \sum_{i} T(O_i) + \phi_{n+1} - \phi_n + \phi_n - \phi_{n-1} + \phi_{n+1} - \phi_0$$

final pot.

initial pot.

Total actual cost: $T(n)$

another: $A(n)$

$$T(n) = A(n) + \phi_n - \phi_{n+1}$$

If $\phi_{n+1} - \phi_0 > 0$ then

$$T(n) \leq A(n)$$
Example: For the case of stacks, let us define
\[
\phi(S) = \# \text{ elements in stack}
\]
\[
\phi(\text{empty stack}) = 0 \quad \phi_k - \phi_0 \geq 0
\]
\[
A(\text{push}) = 1 + 1 = 2
\]
\[
A(\text{pop}) = 1 + (-1) = 0
\]
\[
A(\text{empty stack}) = K - (K) = 0
\]
The maximum amortized cost of any \( op_m \) is 2, so total amortized cost of \( n \) operations \( \leq 2n \)

h.w.: Try to come up with an appropriate potential function for the array-based search data structure
Problem: Given \( n \) strings over some finite alphabet \( \Sigma \), we want to arrange them in lexicographic order.

Strings \( S_1, S_2, \ldots, S_n \) have lengths \( l_1, l_2, \ldots, l_n \),

\[
\sum_{i=1}^{n} l_i = n
\]

Special case: all \( l_i \)'s are equal

Run radix sort