طراحی الگوریتم ها

مبحث دهم: تحلیل سرشکن

سجاد شیرعلی شهرضا

بهار 1402

یک شنبه، 21 اسفند 1401

اطلاع رسانی

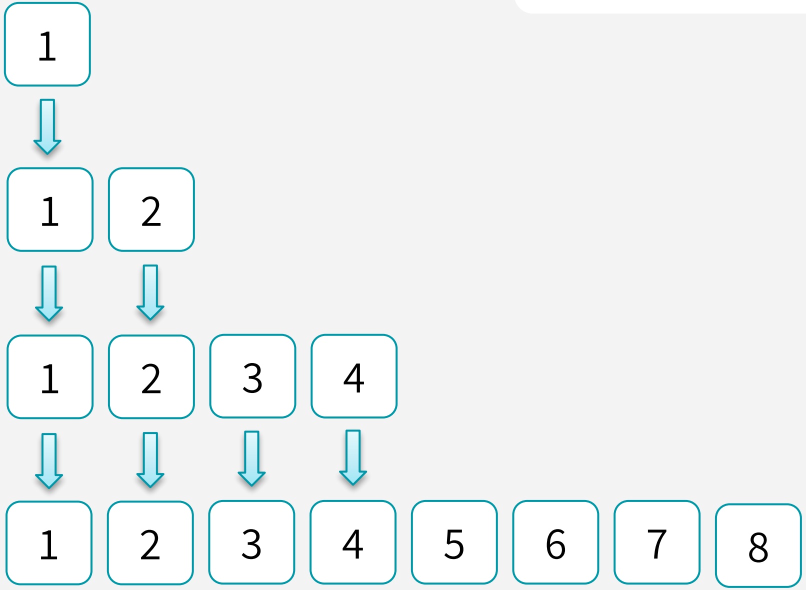
- بخش مرتبط کتاب برای این جلسه: 17
- برگزاری جلسه سه شنبه به صورت مجازی

آرایه پویا

افزایش پویای اندازه آرایه در طول زمان

DYNAMIC ARRAY

We fill it with n elements. When it is FULL, we replaced it with a new array that has 2*n capacity.



1

1 2

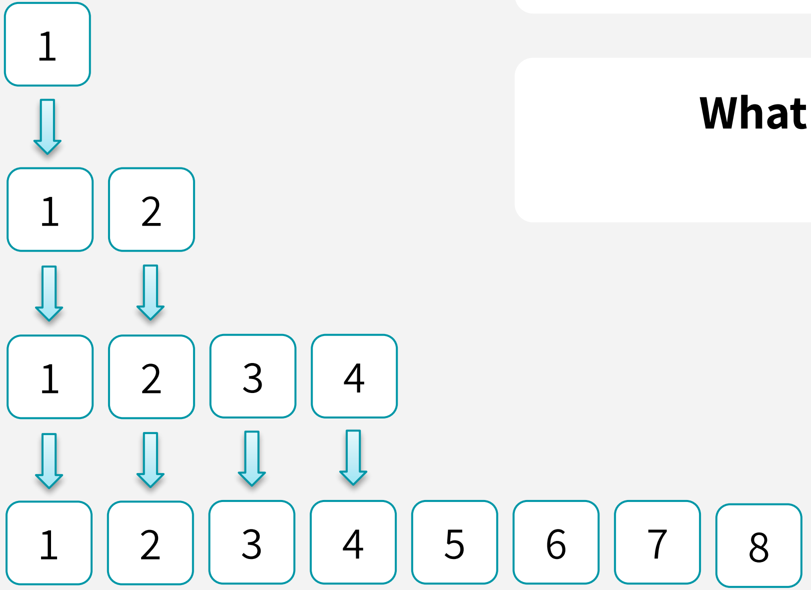
1 2 3 4

1 2 3 4 5 6 7 8

DYNAMIC ARRAY

We fill it with n elements. When it is FULL, we replaced it with a new array that has 2*n capacity.

What is the cost of EACH INSERTION?
What is the WORST CASE?



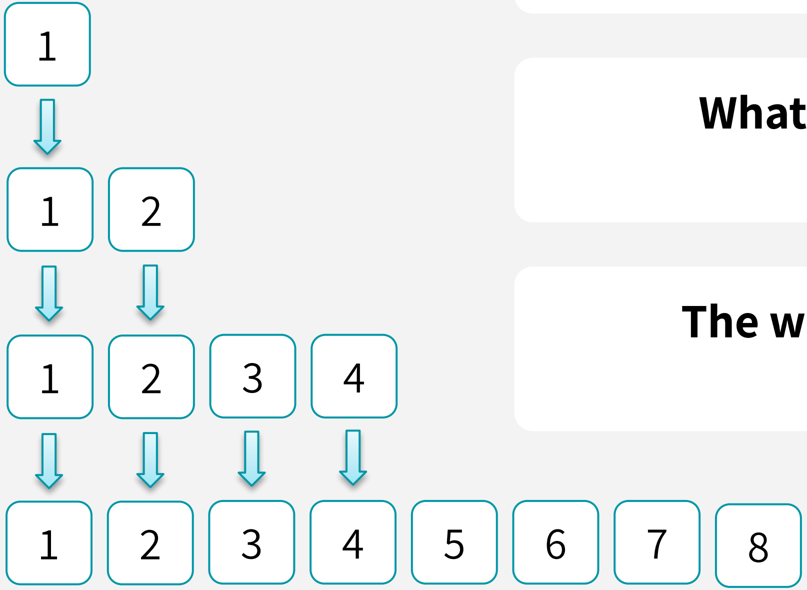
1

1 2

1 2 3 4

1 2 3 4 5 6 7 8

DYNAMIC ARRAY



We fill it with n elements. When it is FULL, we replaced it with a new array that has 2*n capacity.

What is the cost of **EACH INSERTION**?
What is the the **WORST CASE**?

The worst insertion doubles the array!
So, In worst case **O(n)**?

DYNAMIC ARRAY

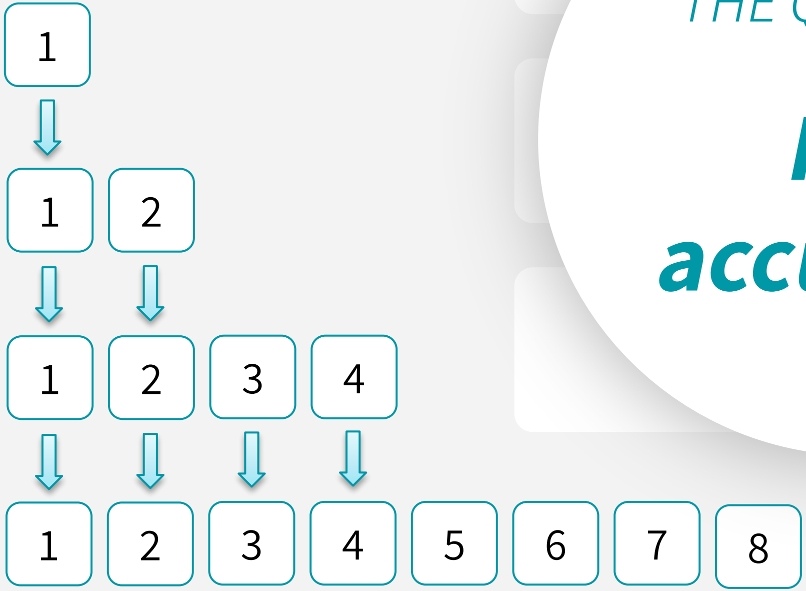
W nts. When it is FULL, we
re that has 2*n capacity.

H INSERTION?
T CASE?

oubles the array!
st case O(n)?

THE QUESTION IS...

Is it accurate?

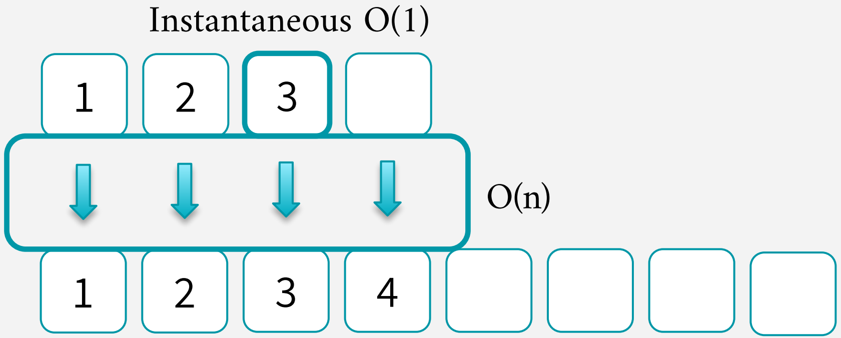


ANALYZING TIME COMPLEXITY

- Two type of operations
  - Simple operations with O(1)
  - Complex operations with O(n)

We need a new type of analysing:

Amortized Analysis

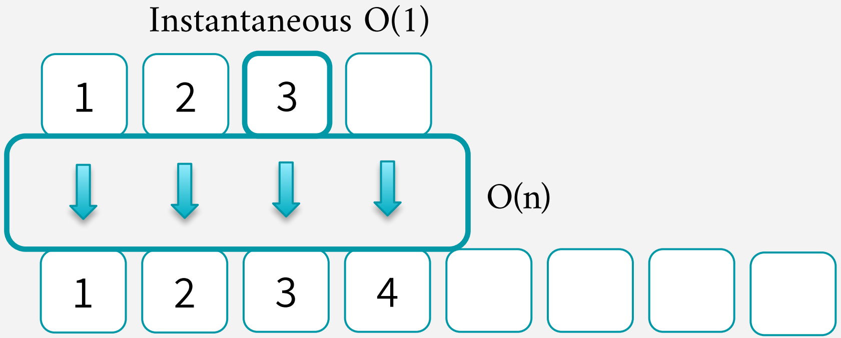


ANALYZING TIME COMPLEXITY

- Two type of operations
  - Simple operations with O(1)
  - Complex operations with O(n)

We need a new type of analysing:

Amortized Analysis



سوال؟

تحلیل سرشکن

AMORTIZED ANALYSIS

- Not just consider one operation, but a sequence of operations
- Average cost over a sequence of operations.

- Example: Dynamic Array

AMORTIZED vs. PROBABILISTIC

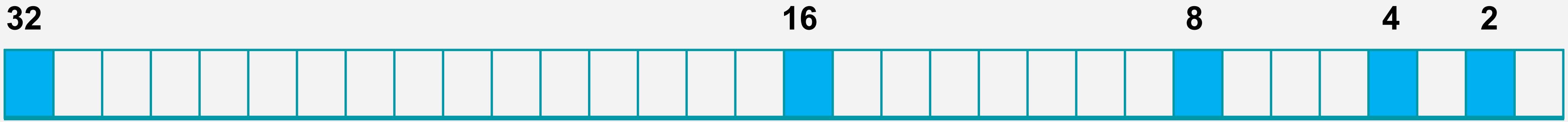
- **Probabilistic analysis:**
  - Average case running time: average over all possible inputs for one algorithm (operation)
  - If using probability, called Expected Running Time.

- **Amortized analysis:**
  - No involvement of probability
  - Average performance on a sequence of operations
  - **Guarantee average performance of each operation among the sequence in worst case**

AMORTIZED ANALYSIS METHODS

- **Aggregate analysis**:
  - Total cost of n operations/n,
- **Accounting Method**:
  - Pay extra credit in each operation
  - Save extra credit on elements
  - Use extra credit for expensive operations
- **Potential method**:
  - Same as accounting method
  - But store the credit in one place as **potential energy**

EXPENSIVE INSERT OPERATION



□ = 1

■ = ?

Becomes more expensive, but happens less frequently

INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

What is the simplest way to determine the cost of each INSERTION?

INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

What is the simplest way to determine the cost of each INSERTION?

Average

سوال؟

روش انبوهه

میانگین هزینه های یک سری عملیات

AGGREGATE ANALYSIS

The **simplest** way to perform amortized analysis

How to calculate? $\frac{\text{Total cost}}{\text{\# of operations}}$

$$O\left(\sum Cost\ of\ n\ operations\right) = O\left(\begin{array}{c}\sum Cost\ of\ \boldsymbol{Cheap}\ operations \\ + \\ \sum Cost\ of\ \boldsymbol{Expensive}\ operations\end{array}\right)$$

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

After inserting 1 items, total cost of expensive insertions = 1

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

After inserting 2 items, total cost of expensive insertions = 3

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

Upto inserting 3 items, total cost of expensive insertions = 3

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

After inserting 4 items, total cost of expensive insertions = 7

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

Upto inserting 7 items, total cost of expensive insertions = 7

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

After inserting 8 items, total cost of expensive insertions = 15

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

Upto inserting 15 items, total cost of expensive insertions = 15

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

After inserting 16 items, total cost of expensive insertions = 31

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

Upto inserting 31 items, total cost of expensive insertions = 31

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
16	1	1	1	1	1	1	1	8	1	1	1	4	1	2	1	Cost
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
32	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Cost

After inserting 32 items, total cost of expensive insertions = 63

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
31	15	15	15	15	15	15	15	15	7	7	7	7	3	3	1	Total Exp
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
63	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	Total Exp

Relation between item # and total cost of expensive operations?

TOTAL EXPENSIVE INSERTION COST

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Item
31	15	15	15	15	15	15	15	15	7	7	7	7	3	3	1	Total Exp
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	Item
63	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	Total Exp

2 x item # > total cost of expensive

AGGREGATE ANALYSIS

The **simplest** way to perform amortized analysis

How to calculate? $\frac{\text{Total cost}}{\text{\# of operations}}$

$$O\left(\sum Cost\ of\ n\ operations\right) = O\left(\begin{array}{c} \sum Cost\ of\ \boldsymbol{Cheap}\ operations \\ + \\ \sum Cost\ of\ \boldsymbol{Expensive}\ operations \end{array}\right)$$

AGGREGATE ANALYSIS

The **simplest** way to perform amortized analysis

How to calculate? $\dfrac{\text{Total cost}}{\text{\# of operations}}$

$$o\left(\sum \mathit{Cost\ of\ n\ operations}\right) = O\left(\begin{array}{c} \sum \mathit{Cost\ of}\ \boldsymbol{Cheap}\ \mathit{operations} \\ + \\ \sum \mathit{Cost\ of}\ \boldsymbol{Expensive}\ \mathit{operations} \end{array}\right)$$

Total cost of n **cheap** operations < n = **O(n)**

AGGREGATE ANALYSIS

The **simplest** way to perform amortized analysis

How to calculate? $\frac{\text{Total cost}}{\text{\# of operations}}$

$$o\left(\sum \mathit{Cost\ of\ n\ operations}\right) = O\left(\begin{matrix} \sum \mathit{Cost\ of}\ \boldsymbol{Cheap}\ \mathit{operations} \\ + \\ \sum \mathit{Cost\ of}\ \boldsymbol{Expensive}\ \mathit{operations} \end{matrix}\right)$$

Total cost of n **cheap** operations < n = **O(n)**
Total cost of **expensive** operations < 2 x n = **O(n)**

AGGREGATE ANALYSIS

The **simplest** way to perform amortized analysis

How to calculate? $\dfrac{\text{Total cost}}{\text{\# of operations}}$

$$O\left(\sum \mathit{Cost\ of\ n\ operations}\right) = O\left(\begin{array}{c} \sum \mathit{Cost\ of}\ \boldsymbol{Cheap}\ \mathit{operations} \\ + \\ \sum \mathit{Cost\ of}\ \boldsymbol{Expensive}\ \mathit{operations} \end{array}\right)$$

Total cost of n **cheap** operations < n = **O(n)**

Total cost of **expensive** operations < 2 x n = **O(n)**

Total cost < **O(n) + O(n) = O(n)**

AGGREGATE ANALYSIS

Dynamic array insertion cost

$$\text{Amortized cost} = \frac{\text{Total cost}}{\text{\# of operations}} = O(n) / n = O(1)$$

سوال؟

روش حسابداری

جمع آوری هزینه اضافه در حین انجام عملیات ساده

ACCOUNTING METHOD

- Save your money for a rainy day!
- Assign every operation a **cost**
  - Use part of it for the operation
  - Save surplus **beside** new item
- **Cheap** operations will have **extra** cost
  - Will help to afford **Expensive** operations
- **Challenge**: Bank balance must always be **0 or positive**

DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

0$

DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

2$

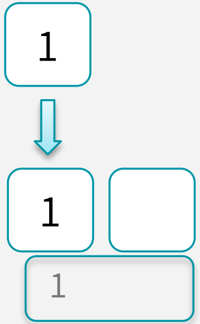
1

DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

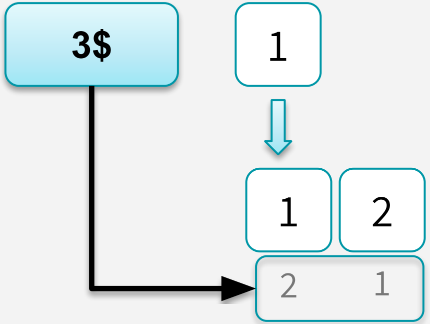
1$



DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

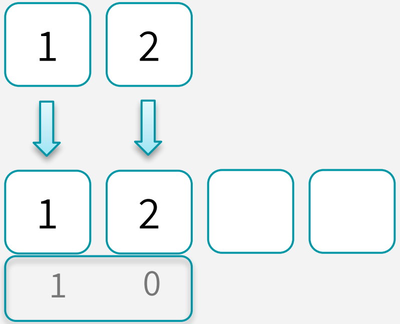


DYNAMIC ARRAY

Total Credit

1$

Charge 3 units per operation

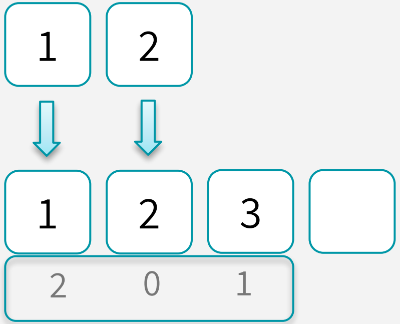


DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

3$

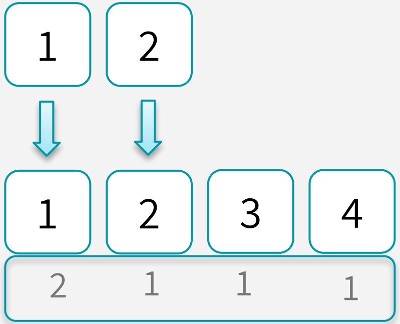


DYNAMIC ARRAY

Total Credit

5$

Charge 3 units per operation


1 2

1 2 3 4

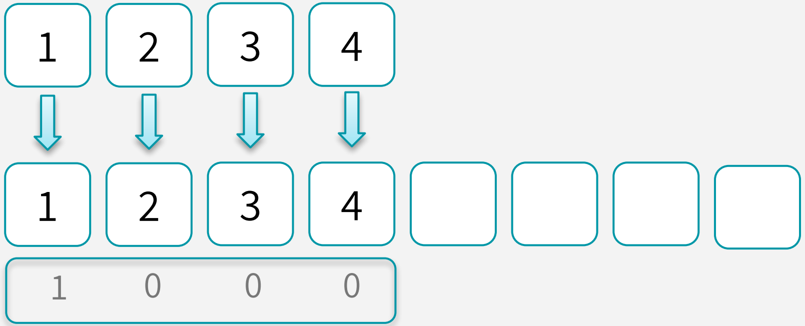
2 1 1 1

DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

1$

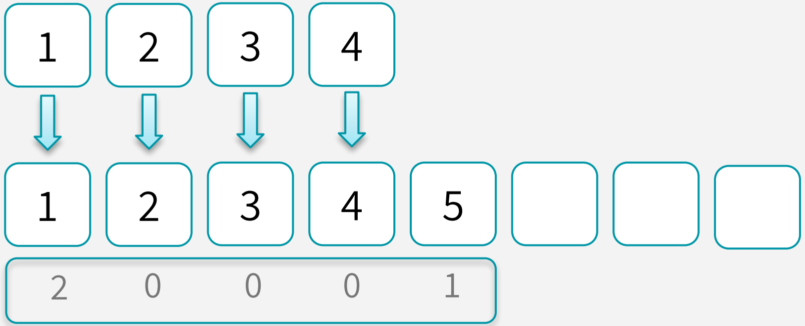


DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

3$

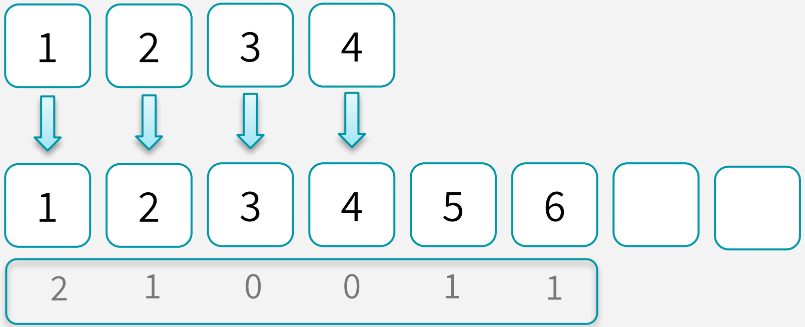


DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

5$

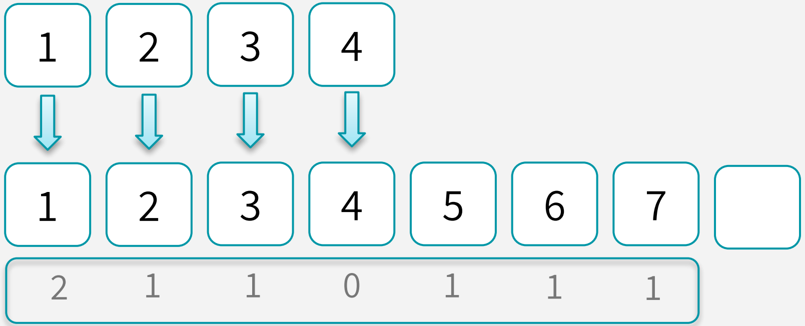


DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

7$



1 2 3 4

1 2 3 4 5 6 7

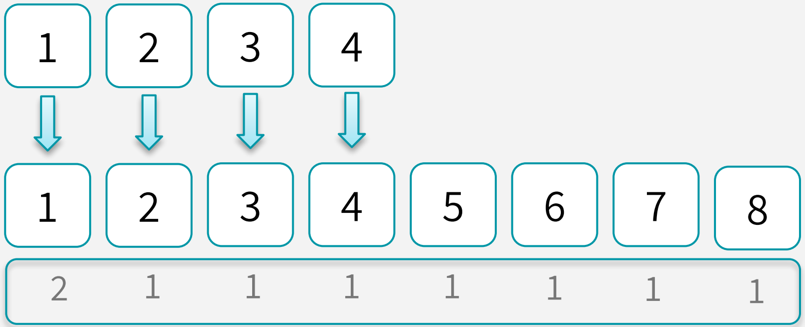
2 1 1 0 1 1 1

DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

9$

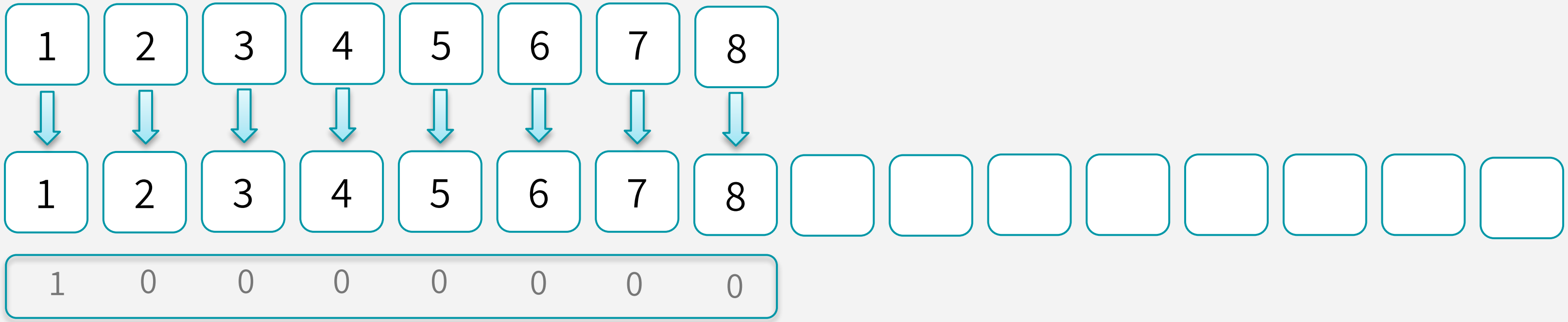


DYNAMIC ARRAY

Charge 3 units per operation

Total Credit

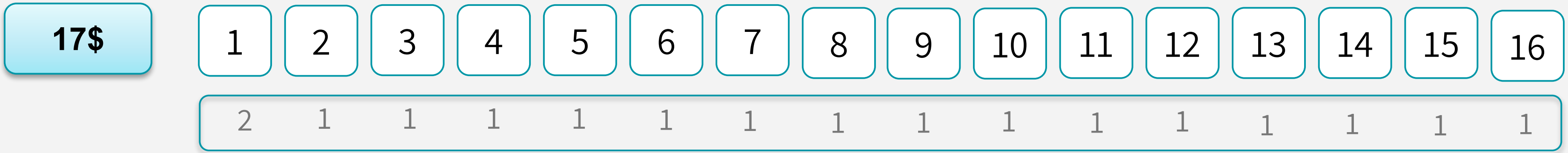
1$



DYNAMIC ARRAY

Charge 3 units per operation

Total Credit



17$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Each operation costs 3, i.e., O(1)
Amortized cost = O(1)

سوال؟

روش پتانسیل

حالت بسط داده شده ای از روش حسابداری

POTENTIAL METHOD

Same as Accounting method

Pay extra for cheap operations and store extra credit.
Use stored credit for expensive operations.

Different from Accounting method

The prepaid work not as credit,
but as “potential energy”, or “potential”

Potential: associated with the **whole data structure**
Credit: associated with **specific objects** in the data structure

DIFFERENCE FROM ACCOUNTING

In Accounting method, Bank balance of particular state is dependent on previous state

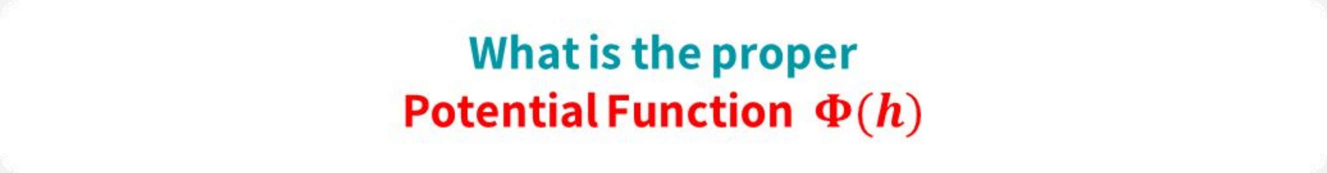
Potential Method uses Potential Function $\Phi(h)$

Potential function:
independently derive the potential at any state

Can compute the **potential difference**:
The change in cost between two operations

POTENTIAL FUNCTION

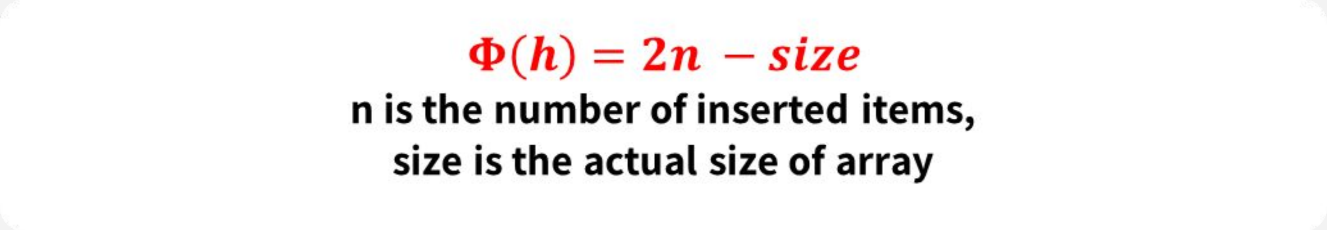
Big challenge



What is the proper

Potential Function $\Phi(h)$

Example: Dynamic array



$$\Phi(h) = 2n - size$$

n is the number of inserted items,

size is the actual size of array

Potential function: must always be non-negative

DYNAMIC ARRAY (n=1)

Energy
~~Bank~~

0

1	

$$\Phi(h) = 2n - size \quad = 2*1 - 2 = 0$$

DYNAMIC ARRAY (n=2)

Energy

0

1	2		

$\Phi(h) = 2n - size$ = 2*2 - 4 = 0

DYNAMIC ARRAY (n=3)

Energy

2

1	2	3	

$\Phi(h) = 2n - size$ = 2*3 - 4 = 2

DYNAMIC ARRAY (n=4)

Energy

0

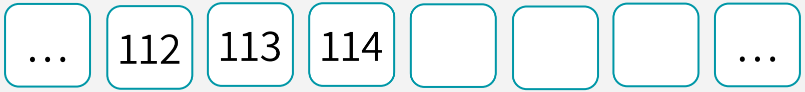
1	2	3	4				

$\Phi(h) = 2n - size$ = 2*4 - 8 = 0

DYNAMIC ARRAY (n=114)

Energy

100



…	112	113	114				…

$$\Phi(h) = 2n - size = 2*114 - 128 = 100$$

POTENTIAL FUNCTION

Amortized cost of the **i**th insertion h_i

$$c_i + \Phi(h_i) - \Phi(h_{i-1})$$

Cost of the **i**th insertion

Potential difference of **i**th and **i-1**th state

POTENTIAL FUNCTION

Amortized cost of the **i**th insertion h_i

$$c_i + \Phi(h_i) - \Phi(h_{i-1})$$

Cost of the **i**th insertion

Potential difference of **i**th and **i-1**th state

Example: Dynamic array

Two cases:
Normal insert
Insert with Expansion

POTENTIAL FUNCTION

Amortized cost of the **Normal insertion in Dynamic Array**

$$c_i + \Phi(h_i) - \Phi(h_{i-1})$$

$$\Phi(h) = 2n - size$$

Normal insertion doesn’t change the size

…	i-1	i	…

$$= c_i + (2i - size) - (2(i-1) - size)$$

$$= 1 + 2i - size - 2i + 2 + size$$

$$= 3$$

POTENTIAL FUNCTION

Amortized cost of the **Expansive insertion in Dynamic Array**

$$c_i + \Phi(h_i) - \Phi(h_{i-1})$$

$$\Phi(h) = 2n - size$$

Expansion insertion change the size

…	i-1	i	…

$$size_i = 2 * size_{i-1}$$

$$size_{i-1} = i$$

Size after expansion

Size before expansion

$$= c_i + (2i - size_i) - (2(i-1) - size_{i-1})$$

$$= c_i + (2i - 2i) - (2(i-1) - i)$$

$$= (i + 1) + 2i - 2i - 2i + 2 + i$$

$$= 3$$

Insert new item and then copy everything

POTENTIAL FUNCTION

Amortized cost of the insertion in **Dynamic Array**

$$c_i + \Phi(h_i) - \Phi(h_{i-1})$$
$$\Phi(h) = 2n - size$$

…	i-1	i	…

Normal
3 operations
(amortized)
O(1)

Amortized Time
O(1)

Expansion
3 operations
(amortized)
O(1)

سوال؟

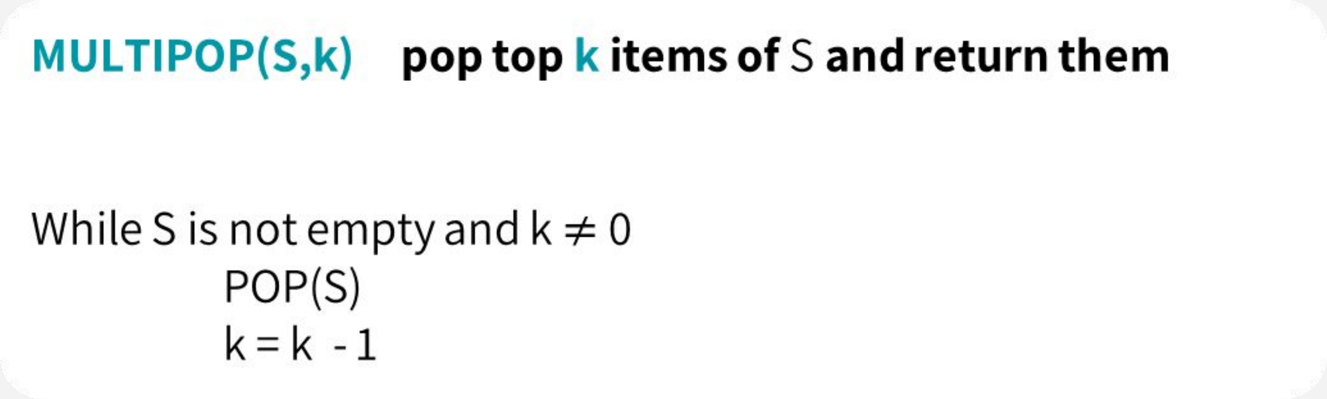
مثال: پشته با حذف چندگانه

حل با استفاده از سه روش معرفی شده

MULTI-POP STACK

PUSH(S,x) **push** x **onto stack** S

POP(S) **pop top item of** S **and return it**



MULTIPOP(S,k) **pop top** **k** **items of** S **and return them**

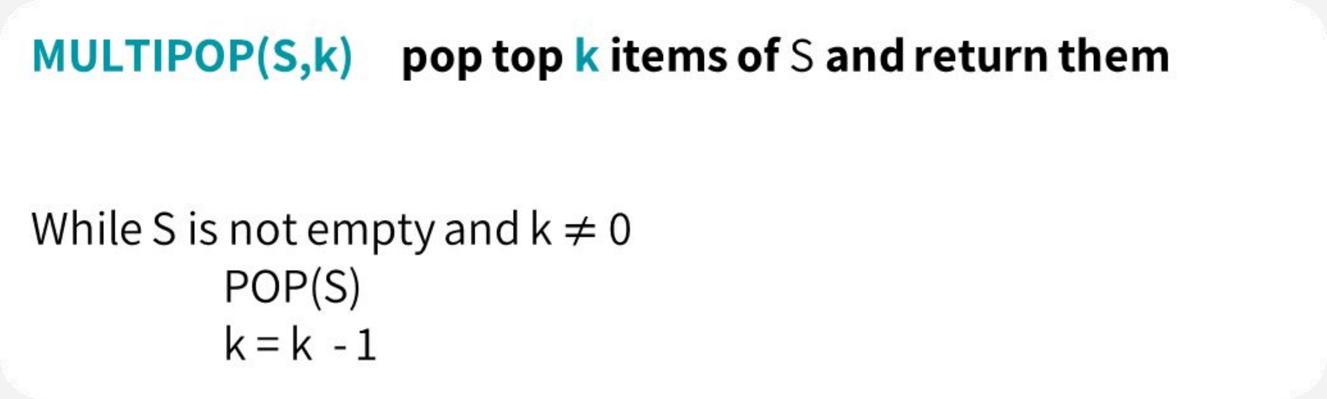
```
While S is not empty and $k \neq 0$
        POP(S)
        $k = k - 1$
```

MULTI-POP STACK

PUSH(S,x) **push** x **onto stack** S

O(1)

POP(S) **pop top item of** S **and return it**



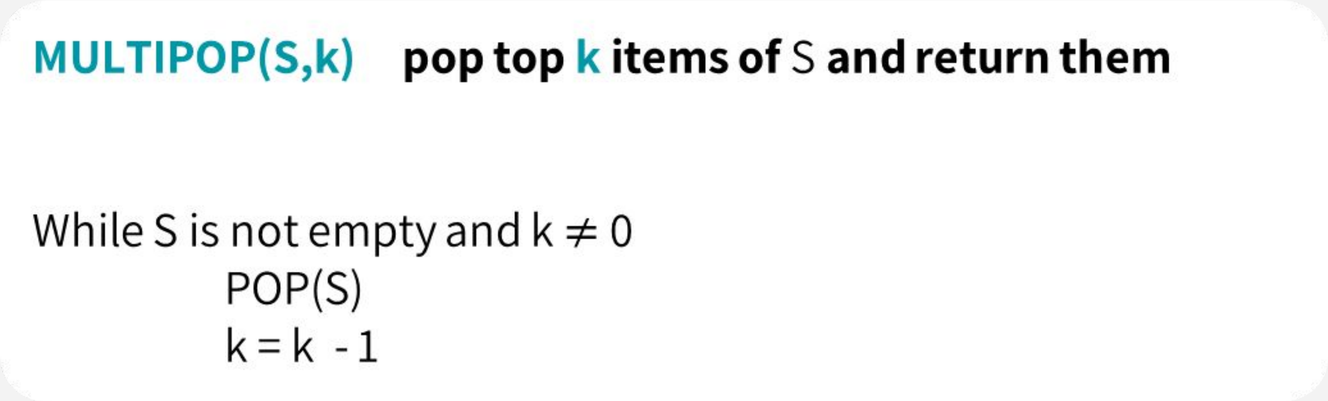
MULTIPOP(S,k) **pop top k items of** S **and return them**

```
While S is not empty and $k \neq 0$
        POP(S)
        $k = k - 1$
```

MULTI-POP STACK

PUSH(S,x) **push** x **onto stack** S — **O(1)**

POP(S) **pop top item of** S **and return it** — **O(1)**



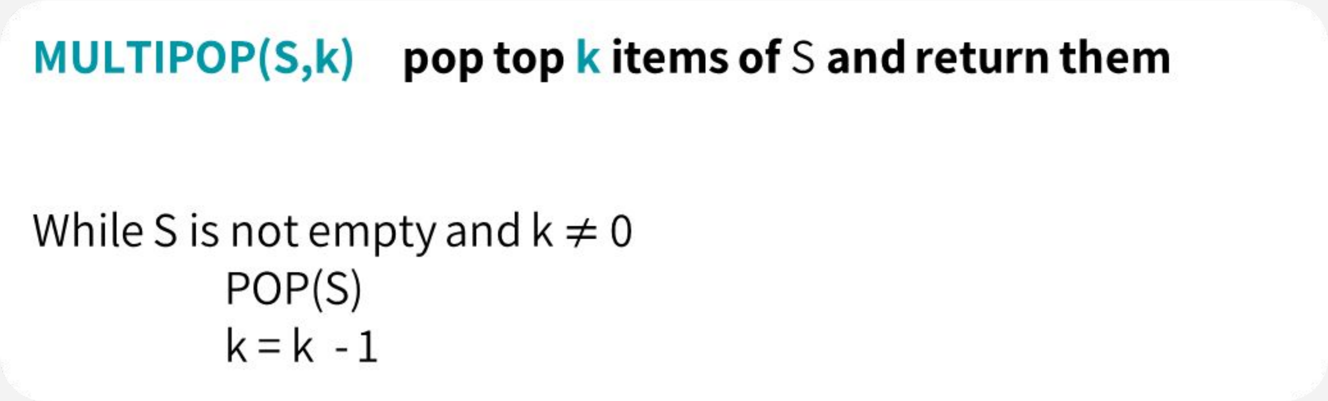
MULTIPOP(S,k) **pop top k items of** S **and return them**

```
While S is not empty and k $\neq$ 0
        POP(S)
        k = k - 1
```

MULTI-POP STACK

PUSH(S,x) **push** x **onto stack** S — **O(1)**

POP(S) **pop top item of** S **and return it** — **O(1)**



MULTIPOP(S,k) **pop top k items of** S **and return them**

```
While S is not empty and k ≠ 0
        POP(S)
        k = k  - 1
```

O(n) ???
θ(n) ???

AGGREGATE ANALYSIS

- Any sequence of n PUSH, POP, and MULTIPOP operations need **O(n)** time
  - At most n items are inserted
  - At most n items are removed
  - In total, at most 2n items are inserted or removed
- Average time per operation is **O(n)**/n = **O(1)**
- Amortized cost = **O(1)**

سوال؟

ACCOUNTING ANALYSIS

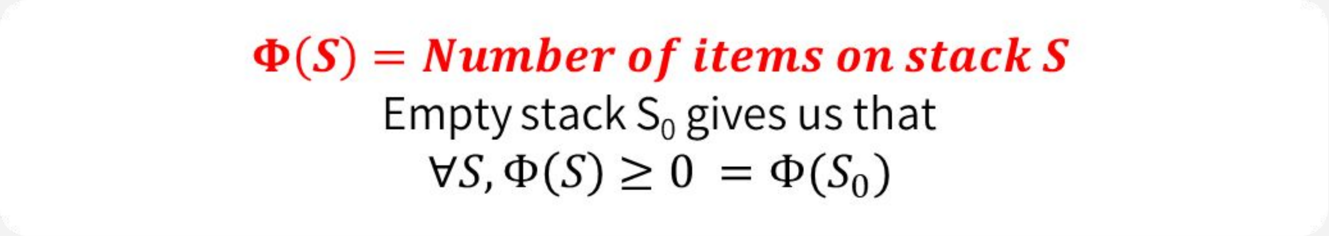
	Real cost	Amortized cost
PUSH(S,x)	1	2
POP(S)	1	0
MULTIPOP(S,k)	Min(k,|s|)	0

Prepay the cost of pop!

Number of pushes = Number of pops

سوال؟

POTENTIAL FUNCTION

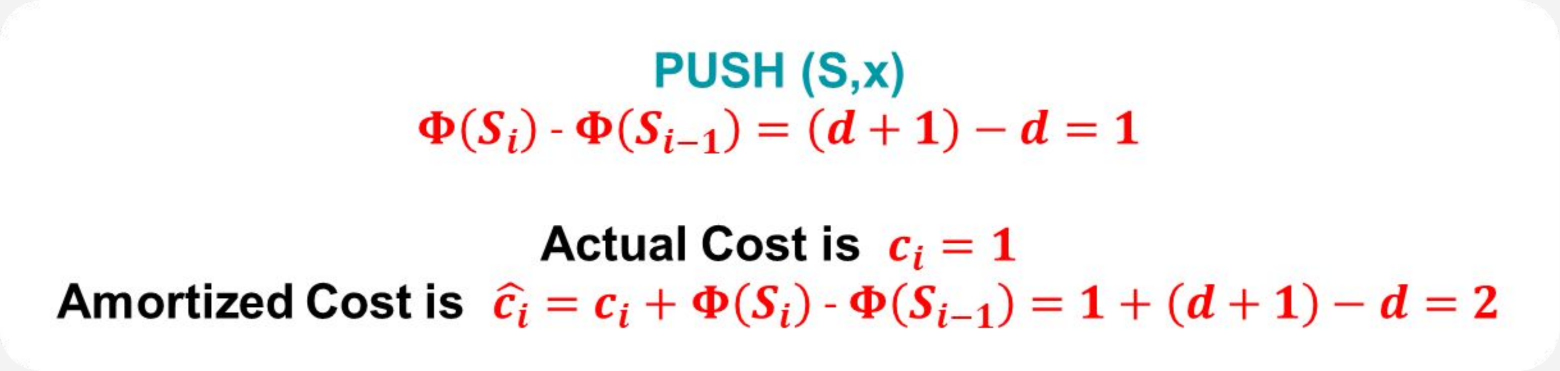


$$\boldsymbol{\Phi(S) = Number\ of\ items\ on\ stack\ S}$$

Empty stack S_0 gives us that

$$\forall S, \Phi(S) \geq 0 = \Phi(S_0)$$

Assume that S_{i-1} has d items on the stack, So:



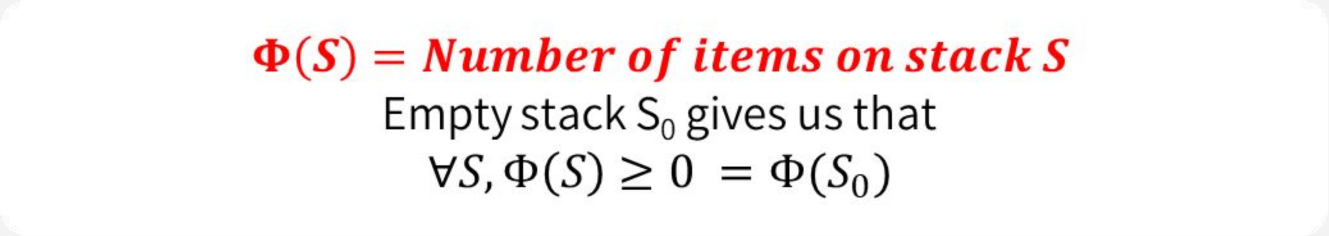
PUSH (S,x)

$$\boldsymbol{\Phi(S_i) - \Phi(S_{i-1}) = (d+1) - d = 1}$$

Actual Cost is $c_i = 1$

Amortized Cost is $\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1}) = 1 + (d+1) - d = 2$

POTENTIAL FUNCTION

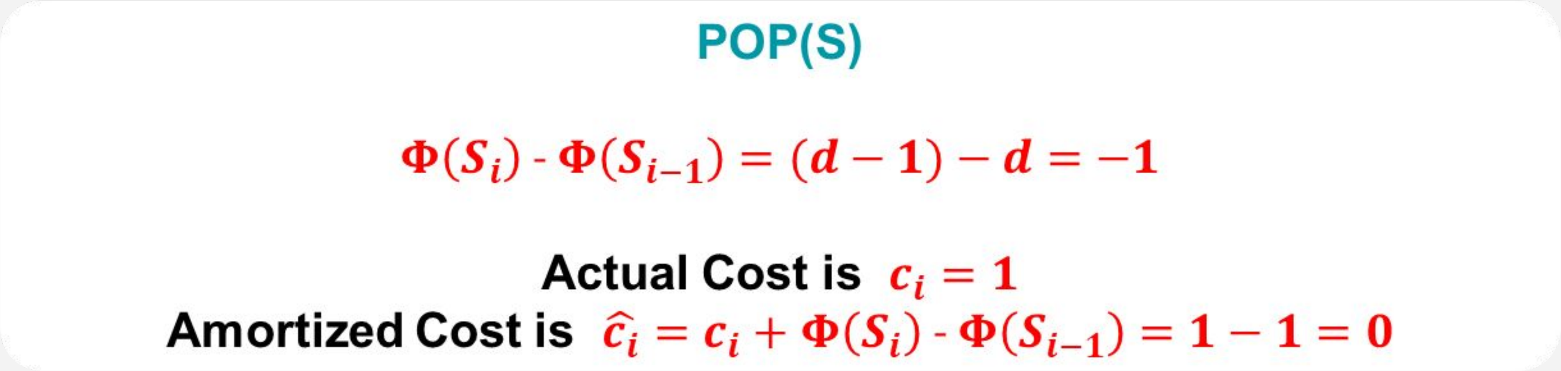


$$\boldsymbol{\Phi(S) = Number\ of\ items\ on\ stack\ S}$$

Empty stack S_0 gives us that

$$\forall S, \Phi(S) \geq 0 = \Phi(S_0)$$

Assume that S_{i-1} has d items on the stack, So:



POP(S)

$$\boldsymbol{\Phi(S_i) \text{ - } \Phi(S_{i-1}) = (d-1) - d = -1}$$

Actual Cost is $c_i = 1$

Amortized Cost is $\hat{c}_i = c_i + \Phi(S_i) \text{ - } \Phi(S_{i-1}) = 1 - 1 = 0$

POTENTIAL FUNCTION

$$\boldsymbol{\Phi(S) = Number\ of\ items\ on\ stack\ S}$$

Empty stack S$_0$ gives us that

$$\forall S, \Phi(S) \geq 0 \ = \Phi(S_0)$$

Assume that S$_{i-1}$ has d items on the stack, So:

MULTIPOP(S)

$$\boldsymbol{k' = min\ (k, d)}$$

$$\boldsymbol{\Phi(S_i)\text{ - }\Phi(S_{i-1}) = (d - k') - d = -k'}$$

Actual Cost is $\boldsymbol{c_i = k'}$

Amortized Cost is $\boldsymbol{\widehat{c_i} = c_i + \Phi(S_i)\text{ - }\Phi(S_{i-1}) = k' - k' = 0}$

POTENTIAL FUNCTION

	Real cost	Amortized cost
PUSH(S,x)	1	2
POP(S)	1	0
MULTIPOP(S,k)	Min(k,|s|)	0

Same as Accounting method

سوال؟

مثال: شمارنده بیتی

حل با استفاده از سه روش معرفی شده

BINARY COUNTER

Binary Counter

```
INCREMENT(A)
    i ← 0
    while i < length[A] and A[i] = 1 do
        A[i] ← 0
        i ← i + 1
    if i < length[A] then
        A[i] ← 1
```

A[7] A[6] A[5] A[4] A[3] A[2] A[1] A[0]

digit	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	1
6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	1	0	0	0
9	0	0	0	0	1	0	0	1

EXAMPLE

digit	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	cost	Total cost
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	**1**	1	1
2	0	0	0	0	0	0	**1**	0	2	3
3	0	0	0	0	0	0	**1**	**1**	1	4
4	0	0	0	0	0	**1**	0	0	3	7
5	0	0	0	0	0	**1**	0	**1**	1	8
6	0	0	0	0	0	**1**	**1**	0	2	10
7	0	0	0	0	0	**1**	**1**	**1**	1	11
8	0	0	0	0	**1**	0	0	0	4	15
9	0	0	0	0	**1**	0	0	**1**	1	16

EXAMPLE

digit	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	cost	Total cost
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	**1**	1	1
2	0	0	0	0	0	0	**1**	0	2	3
3	0	0	0	0	0	0	**1**	**1**	1	4
4	0	0	0	0	0	**1**	0	0	3	7
5	0	0	0	0	0	**1**	0	**1**	1	8
6	0	0	0	0	0	**1**	**1**	0	2	10
7	0	0	0	0	0	**1**	**1**	**1**	1	11
8	0	0	0	0	**1**	0	0	0	4	15
9	0	0	0	0	**1**	0	0	**1**	1	16

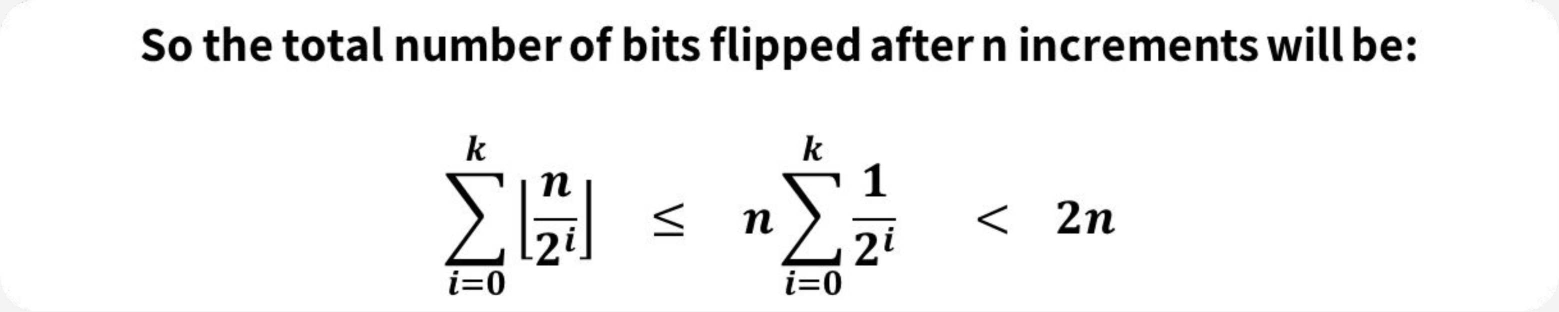
EXAMPLE

Each A[i] flipped after 2^i increments

digit	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	cost	Total cost
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	**1**	1	1
2	0	0	0	0	0	0	**1**	0	2	3
3	0	0	0	0	0	0	**1**	**1**	1	4
4	0	0	0	0	0	**1**	0	0	3	7
5	0	0	0	0	0	**1**	0	**1**	1	8
6	0	0	0	0	0	**1**	**1**	0	2	10
7	0	0	0	0	0	**1**	**1**	**1**	1	11
8	0	0	0	0	**1**	0	0	0	4	15
9	0	0	0	0	**1**	0	0	**1**	1	16

AGGREGATE ANALYSIS

Each A[i] flipped after 2^i increments



So the total number of bits flipped after n increments will be:

$$\sum_{i=0}^{k} \left\lfloor \frac{n}{2^i} \right\rfloor \leq n \sum_{i=0}^{k} \frac{1}{2^i} < 2n$$

So, every operation requires at most 2n/n bit flips on average, i.e., has an amortized cost of O(1)

سوال؟

ACCOUNTING ANALYSIS

Binary Counter

```
INCREMENT(A)
$i \leftarrow 0$
while $i < length[A]$ and $A[i] = 1$ do
    $A[i] \leftarrow 0$
    $i \leftarrow i + 1$
if $i < length[A]$ then
    $A[i] \leftarrow 1$
```

Every 1 that is flipped to be a 0 was originally made into a 1 in a previous operation

Every increment flips exactly one 0 to be a 1

ACCOUNTING ANALYSIS

Binary Counter

```
INCREMENT(A)
    $i \leftarrow 0$
    while $i <$ length[A] and $A[i] = 1$ do
        $A[i] \leftarrow 0$
        $i \leftarrow i + 1$
    if $i <$ length[A] then
        $A[i] \leftarrow 1$
```

Real cost	Amortized cost
1	0
1	2

Prepay the cost of flipping back to 0

Every 1 that is flipped to be a 0 was originally made into a 1 in a previous operation
Every increment flips exactly one 0 to be a 1

سوال؟

POTENTIAL FUNCTION

$$\boldsymbol{\Phi(D) = Number\ of\ 1's\ in\ the\ counter}$$

Suppose that the ith increment operation flips t$_i$ 1 bits to 0

let b$_i$ be the number of 1s in the counter after the operation

Actual cost is $c_i \leq \ t_i + 1$

If $\boldsymbol{b_i = 0}$ **then increment totally resets the counter and** $\boldsymbol{b_{i-1} = t_i = k}$

If $\boldsymbol{b_i > 0}$ **then** $\boldsymbol{b_i = b_{i-1} - t_i + 1}$

In both cases $\boldsymbol{b_i} \leq \ \boldsymbol{b_{i-1} - t_i + 1}$ **so**

$$\boldsymbol{\Phi(D_i) - \Phi(D_{i-1}) \leq b_{i-1} - t_i + 1 - b_{i-1} = 1 - t_i}$$

Amortized Cost is $\boldsymbol{\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \leq (t_i + 1) + (1 - t_i) = 2}$

سوال؟

REFERENCES

- https://www.youtube.com/watch?v=T7W5E-5mljc
- http://home.cse.ust.hk/~golin/COMP572/Notes/Amortized.pdf
    - https://home.cse.ust.hk/~golin/
    - Goli,Comp572 Lectures,2006
- http://www-di.inf.puc-rio.br/~laber/Amortized-official.pdf
    - http://www.inf.puc-rio.br/en/teacher/@eduardo-sany-laber
    - Laber, University of Rio de Janeiro,