Searching Lists

Algorithms and Datastructures

Marjahan Begum and Anders Kalhauge

Spring 2017

Outline

Arrays

Expandable arrays

Arrays are by nature of fixed length. How can we make them expandanble and still have direct memory access?

When adding a new element to a full array we could:

1. Copy the array to an array one bigger
2. Copy the array to an array m elements bigger
3. Copy the array to an array of double size

Expandable arrays

Arrays are by nature of fixed length. How can we make them expandanble and still have direct memory access?

When adding a new element to a full array we could:

1. Copy the array to an array one bigger
2. Copy the array to an array m elements bigger
3. Copy the array to an array of double size

Average time for adding an element will be:

1.
2.
3.

Expandable arrays

Arrays are by nature of fixed length. How can we make them expandanble and still have direct memory access?

When adding a new element to a full array we could:

1. Copy the array to an array one bigger
2. Copy the array to an array m elements bigger
3. Copy the array to an array of double size

Average time for adding an element will be:

1. $O(n)$ - makes sense all elements are copied at each insert
2.
3.

Expandable arrays

Arrays are by nature of fixed length. How can we make them expandanble and still have direct memory access?

When adding a new element to a full array we could:

1. Copy the array to an array one bigger
2. Copy the array to an array m elements bigger
3. Copy the array to an array of double size

Average time for adding an element will be:

1. $O(n)$ - makes sense all elements are copied at each insert
2. $O(n)$ - all elements are copied each m^{th} time $O(\frac{n}{m}) = O(n)$
3.

Expandable arrays

Arrays are by nature of fixed length. How can we make them expandanble and still have direct memory access?

When adding a new element to a full array we could:

1. Copy the array to an array one bigger
2. Copy the array to an array m elements bigger
3. Copy the array to an array of double size

Average time for adding an element will be:

1. $O(n)$ - makes sense all elements are copied at each insert
2. $O(n)$ - all elements are copied each m^{th} time $O(\frac{n}{m}) = O(n)$
3. $O(1)$ - how can that be?

Expandable arrays

Money in the bank

Balance: 4 we hope that is enough to pay for future expansions

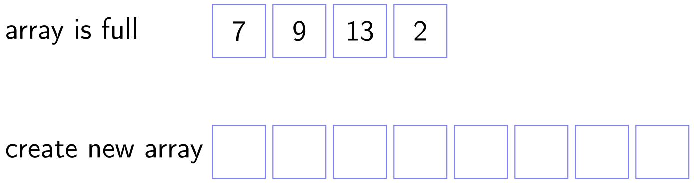
array is full

7	9	13	2

Expandable arrays

Money in the bank

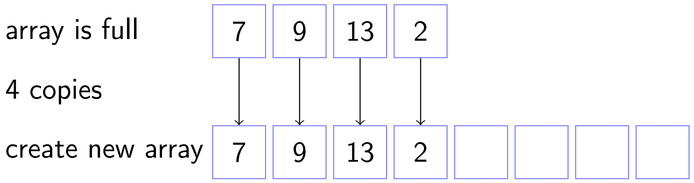
Balance: $4 - 0 = 4$ (creating new array considered free here)



Expandable arrays

Money in the bank

Balance: $4 - 4 = 0$ (using 1 per copy)



Expandable arrays

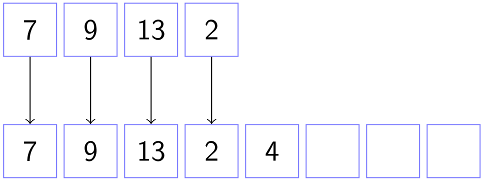
Money in the bank

Balance: $0 + 3 - 1 = 2$ (charging 3 for an insert, using 1)

array is full

4 copies

create new array



add element 4

Expandable arrays

Money in the bank

Balance: $2 + 3 - 1 = 4$ (charging 3 for an insert, using 1)

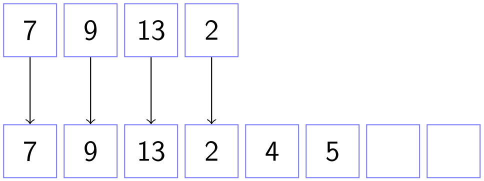
array is full

4 copies

create new array

add element 4

add element 5



Expandable arrays

Money in the bank

Balance: $4 + 3 - 1 = 6$ (charging 3 for an insert, using 1)

array is full

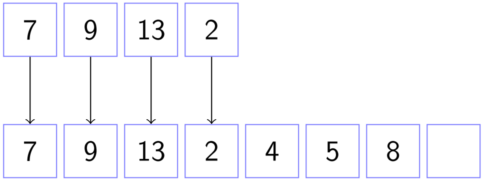
4 copies

create new array

add element 4

add element 5

add element 8



Expandable arrays

Money in the bank

Balance: $6 + 3 - 1 = 8$ (charging 3 for an insert, using 1)

array is full

4 copies

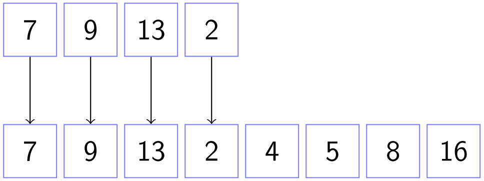
create new array

add element 4

add element 5

add element 8

add element 16



Expandable arrays

Money in the bank

Balance: 8 enough to pay for 8 copies

array is full

4 copies

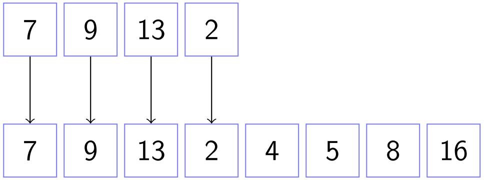
create new array

add element 4

add element 5

add element 8

add element 16



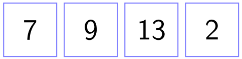
$O(3) = O(1)$

Expandable arrays

Constant payload

Payload: 0

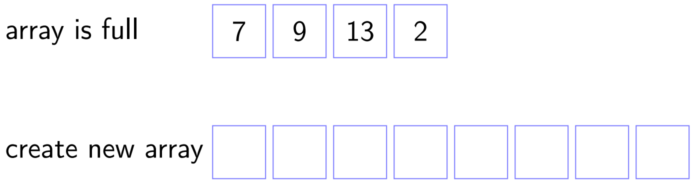
array is full



Expandable arrays

Constant payload

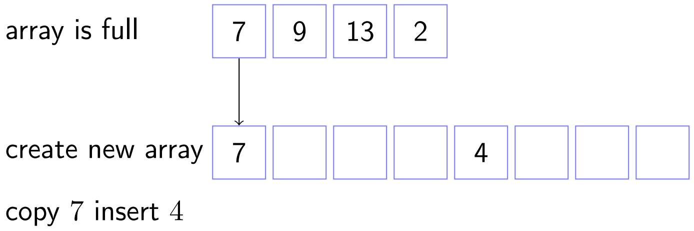
Payload: 0 (creating new array considered free here)



Expandable arrays

Constant payload

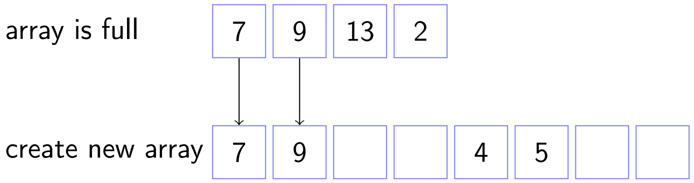
Payload: $1 + 1 = 2$ (1 for copying and 1 for inserting)



Expandable arrays

Constant payload

Payload: $1 + 1 = 2$ (1 for copying and 1 for inserting)



copy 7 insert 4

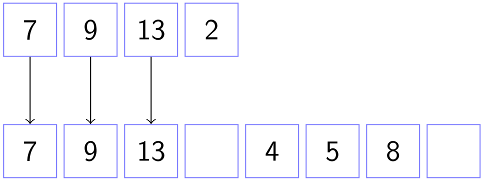
copy 9 insert 5

Expandable arrays

Constant payload

Payload: $1 + 1 = 2$ (1 for copying and 1 for inserting)

array is full



create new array

copy 7 insert 4

copy 9 insert 5

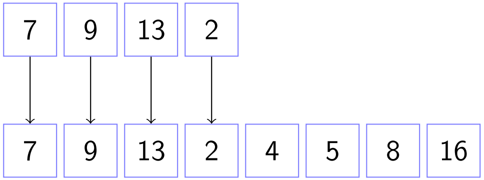
copy 13 insert 8

Expandable arrays

Constant payload

Payload: $1 + 1 = 2$ (1 for copying and 1 for inserting)

array is full



create new array

copy 7 insert 4

copy 9 insert 5

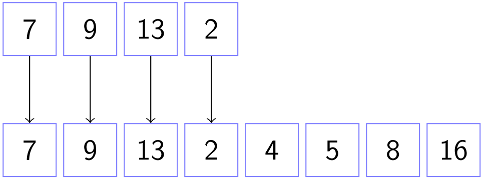
copy 13 insert 8

copy 2 insert 16

Expandable arrays

Constant payload

Payload: 8 in total for 4 insertions



copy 7 insert 4

copy 9 insert 5

copy 13 insert 8

copy 2 insert 16

$$O(2) = O(1)$$

Exercise 1 - Complexity

- What would the complexity (big-O) be if we:
  - Triple the array size instead of doubling it?
  - Only made the new array 50% bigger?
- Bearing in mind that most modern memory is paged[1], consider why doubling the array size is not such a bad idea?

[1]typically in 2^n sized pages

Exercise 2 - Implementation

1. Create a Java class `FlexibleArray` that uses the "Constant payload" algorithm.

   ```
   public class FlexibleArray<T> {
      ...
     public T get(int index) { ... }
     public void set(int index, T element) { ... }
     public void add(T element) { ... }
     public int size() { ...}
     }
   ```

   Note that to create a new array of type `T` you must:

   ```
   private T[] arrayOfT = (T[])new Object[1000];
   ```

2. Measure the time it takes to add 10.000, 100.000, and 1.000.000 elements.
3. Measure Javas build-in `ArrayList` with the same data.