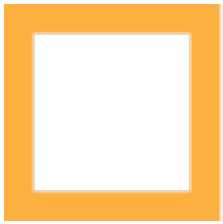
Algorithms and Data Structures
Searching Abstract Data Structures, Symbol Tables

Jacob Trier Frederiksen & Anders Kalhauge



cphbusiness

Spring 2019

Outline

Arrays

Symbol Tables
- List-based
- Array-based

Expandable arrays

Arrays are by nature of fixed length. How can we make them expandable 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 expandable 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 expandable 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

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Arrays are by nature of fixed length. How can we make them expandable 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 expandable 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?

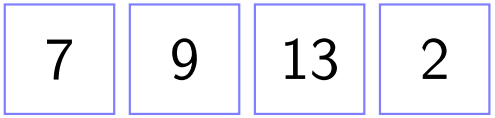
Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Money in the bank

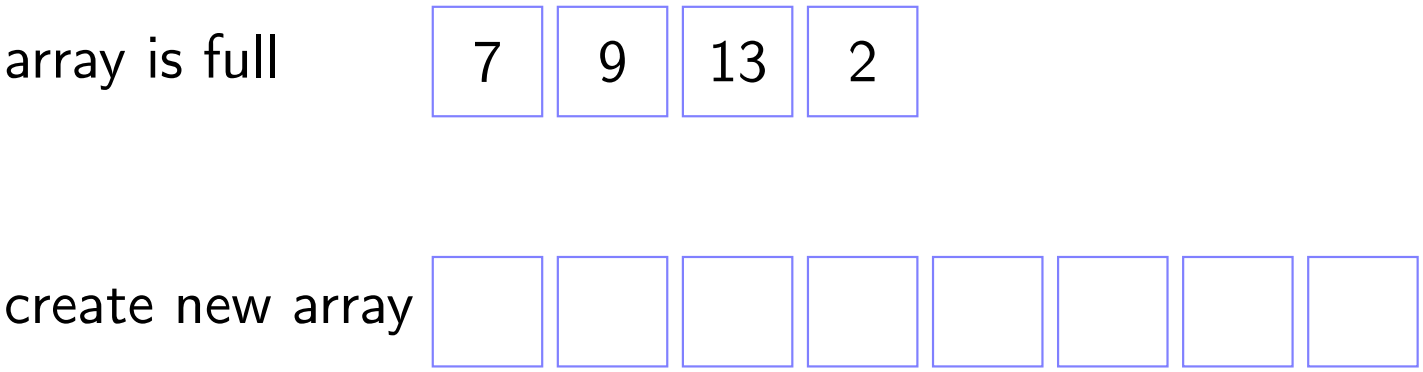
Balance: 4 we hope that is enough to pay for future expansions

array is full



Money in the bank

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

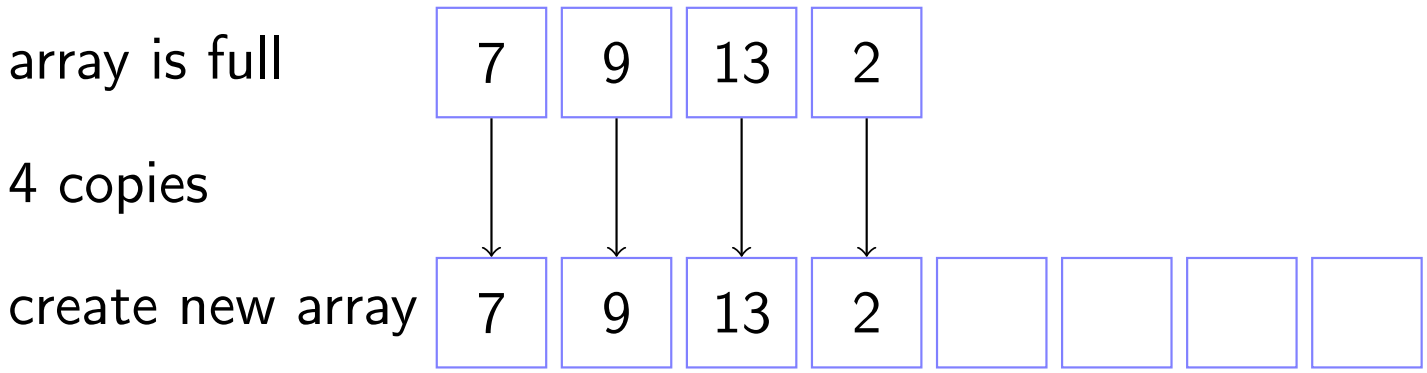


Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Money in the bank

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

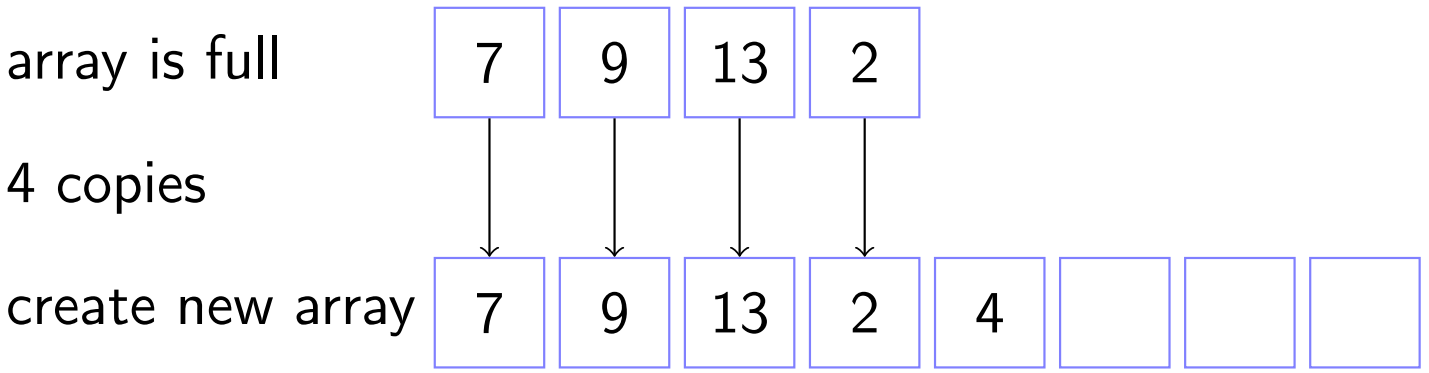


Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Money in the bank

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



add element 4

Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

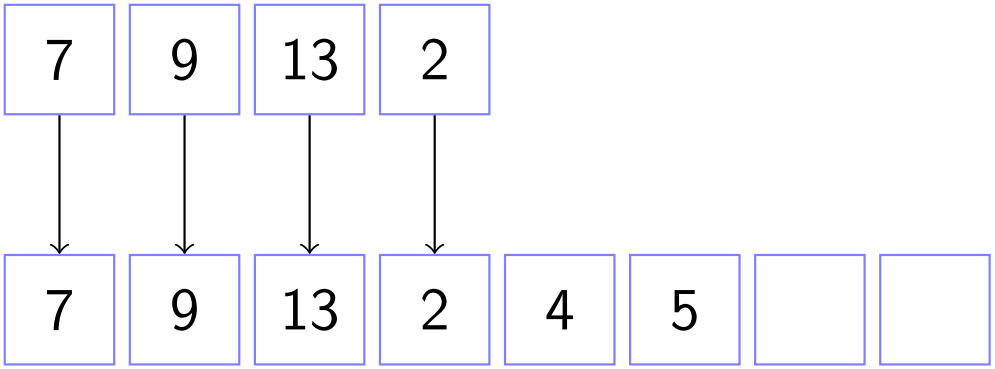
Money in the bank

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

cphbusiness
COPENHAGEN BUSINESS ACADEMY

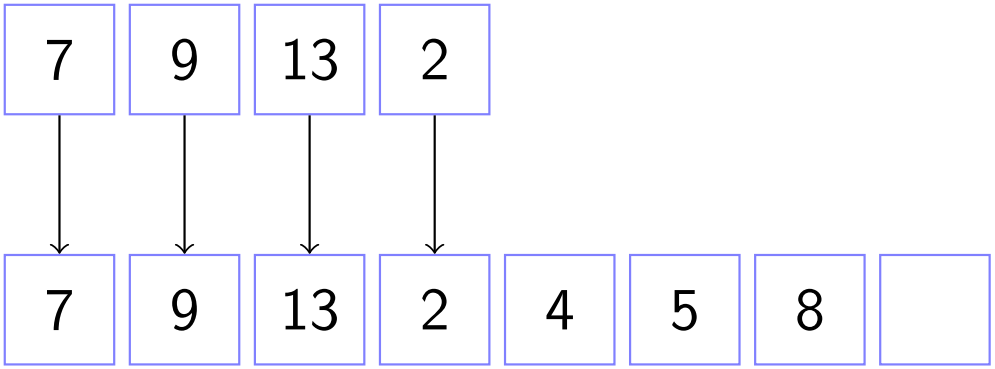
Money in the bank

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

cphbusiness
COPENHAGEN BUSINESS ACADEMY

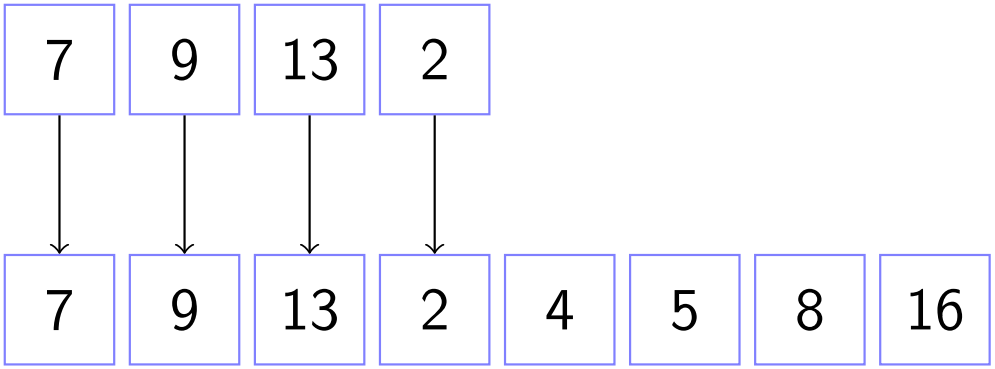
Money in the bank

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

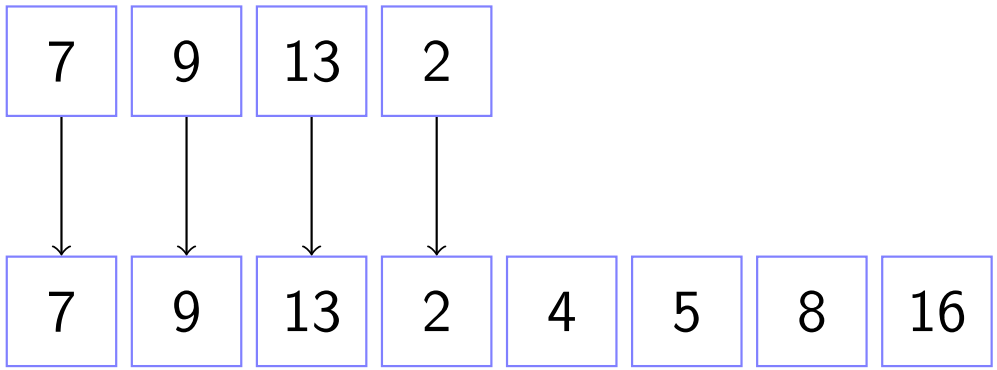
Money in the bank

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

$$O(3) = O(1)$$

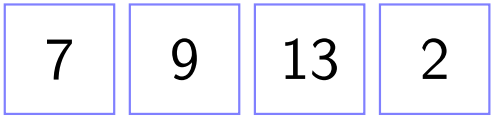
Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Constant payload

Payload: 0

array is full

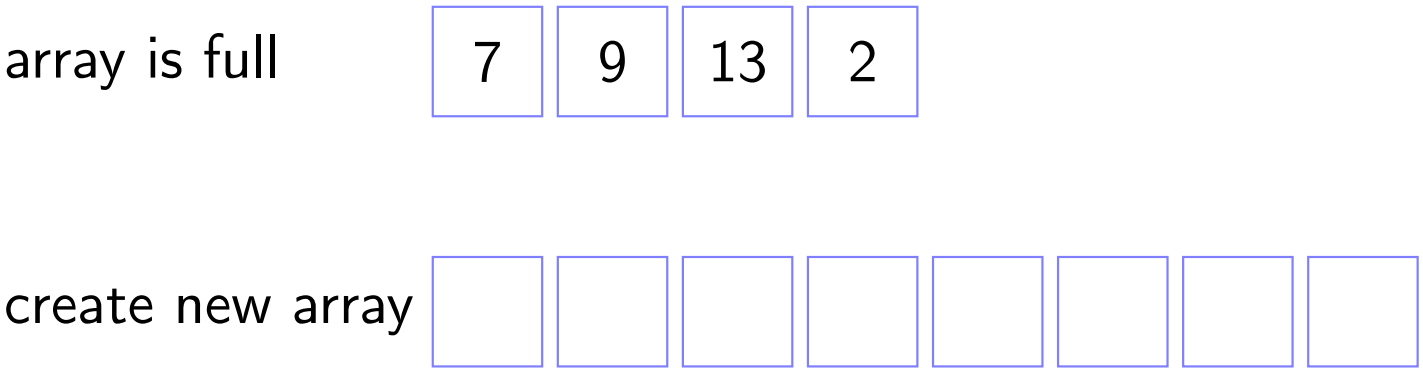


Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Constant payload

Payload: 0 (creating new array considered free here)

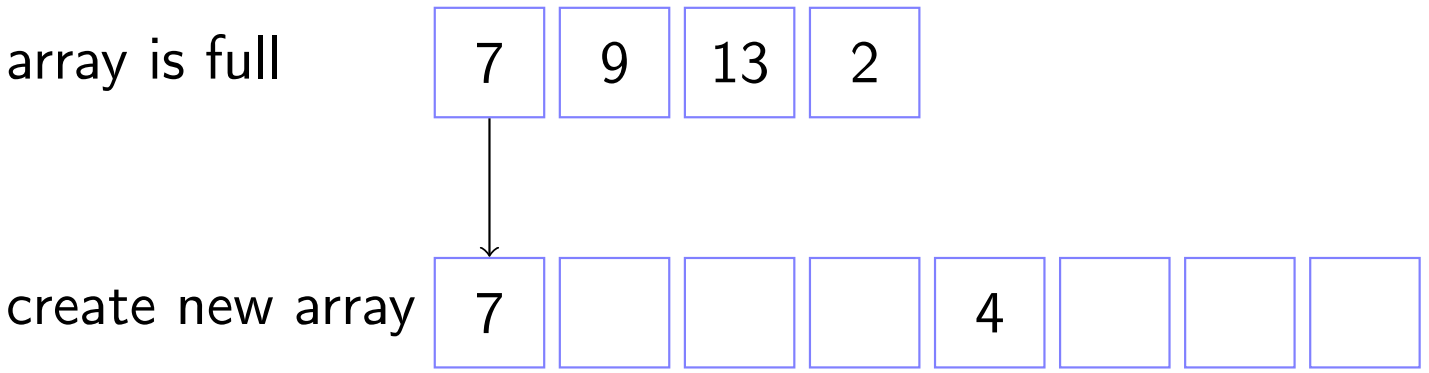


Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Constant payload

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

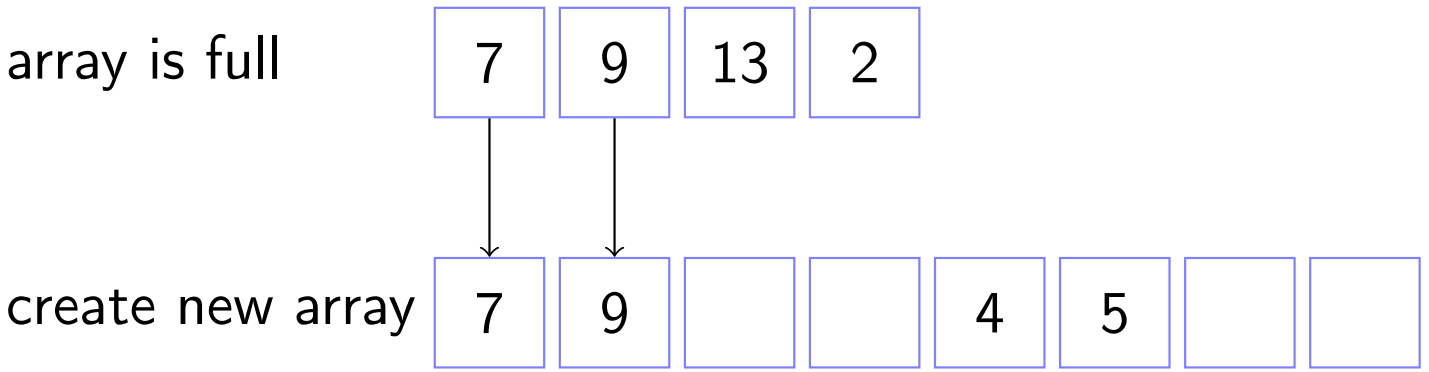


Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Constant payload

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



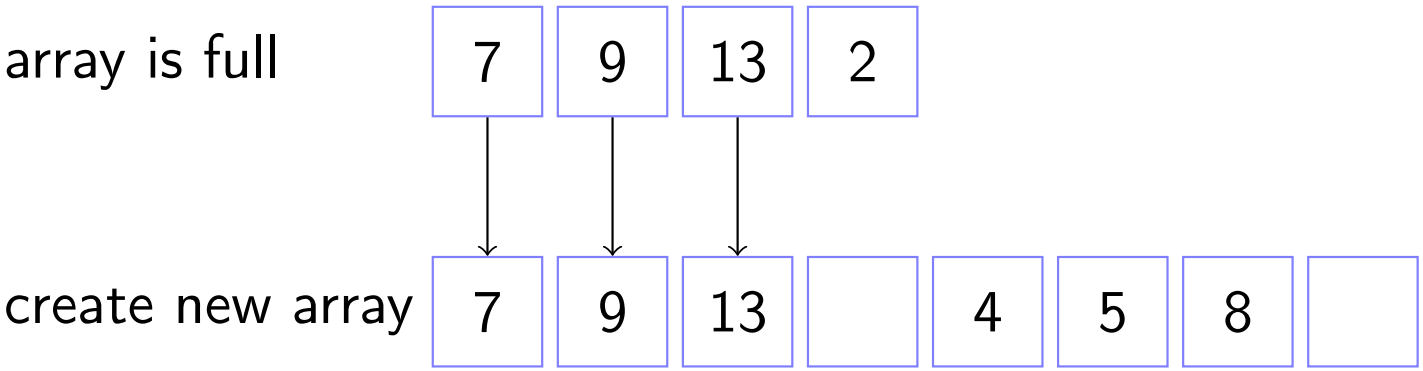
copy 7 insert 4

copy 9 insert 5

Expandable arrays

Constant payload

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



copy 7 insert 4

copy 9 insert 5

copy 13 insert 8

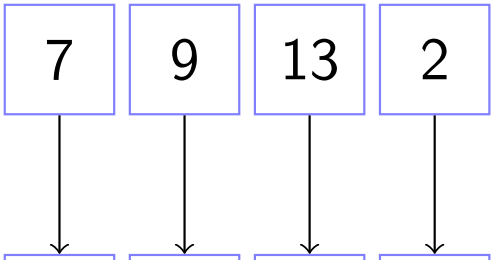
Expandable arrays

cphbusiness
COPENHAGEN BUSINESS ACADEMY

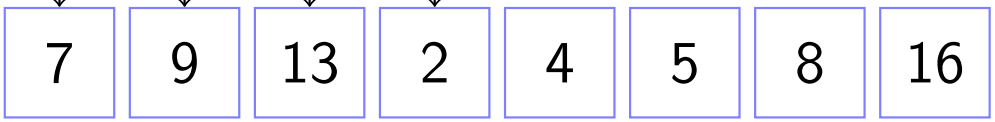
Constant payload

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

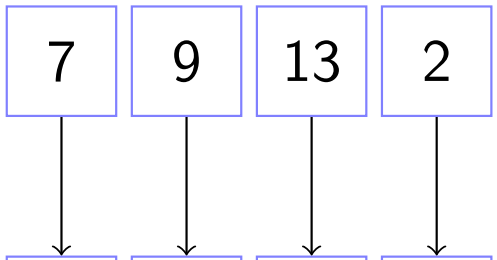
copy 2 insert 16

Expandable arrays

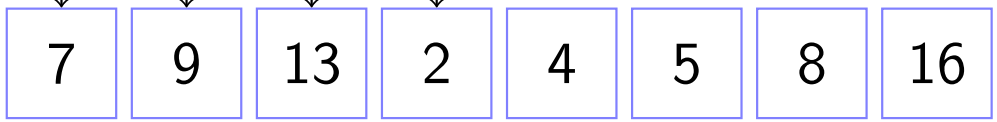
Constant payload

Payload: 8 in total for 4 insertions

array is full



create new array



copy 7 insert 4

copy 9 insert 5

copy 13 insert 8

copy 2 insert 16

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

Resizing Arrays (according to Alg. 4th ed.)

Resizing Arrays - Cost

	best	worst	amortized
construct	1	1	1
push	1	N	1
pop	1	N	1
size	1	1	1

doubling and halving operations

order of growth of running time for resizing stack with N items

Q: discuss the differences from what we just had in slides?

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

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.

Arrays

Symbol Tables

- List-based
- Array-based

Examples of Symbol Tables

application	purpose of search	key	value
dictionary	find definition	word	definition
book index	find relevant pages	term	list of page numbers
file share	find song to download	name of song	computer ID
financial account	process transactions	account number	transaction details
web search	find relevant web pages	keyword	list of page names
compiler	find properties of variables	variable name	type and value
routing table	route Internet packets	destination	best route
DNS	find IP address	domain name	IP address
reverse DNS	find domain name	IP address	domain name
genomics	find markers	DNA string	known positions
file system	find file on disk	filename	location on disk

Symbol tables

`Map` in Java

- Lookup a value based on a key
- No `null` keys
- No `null` values
- No duplicate keys

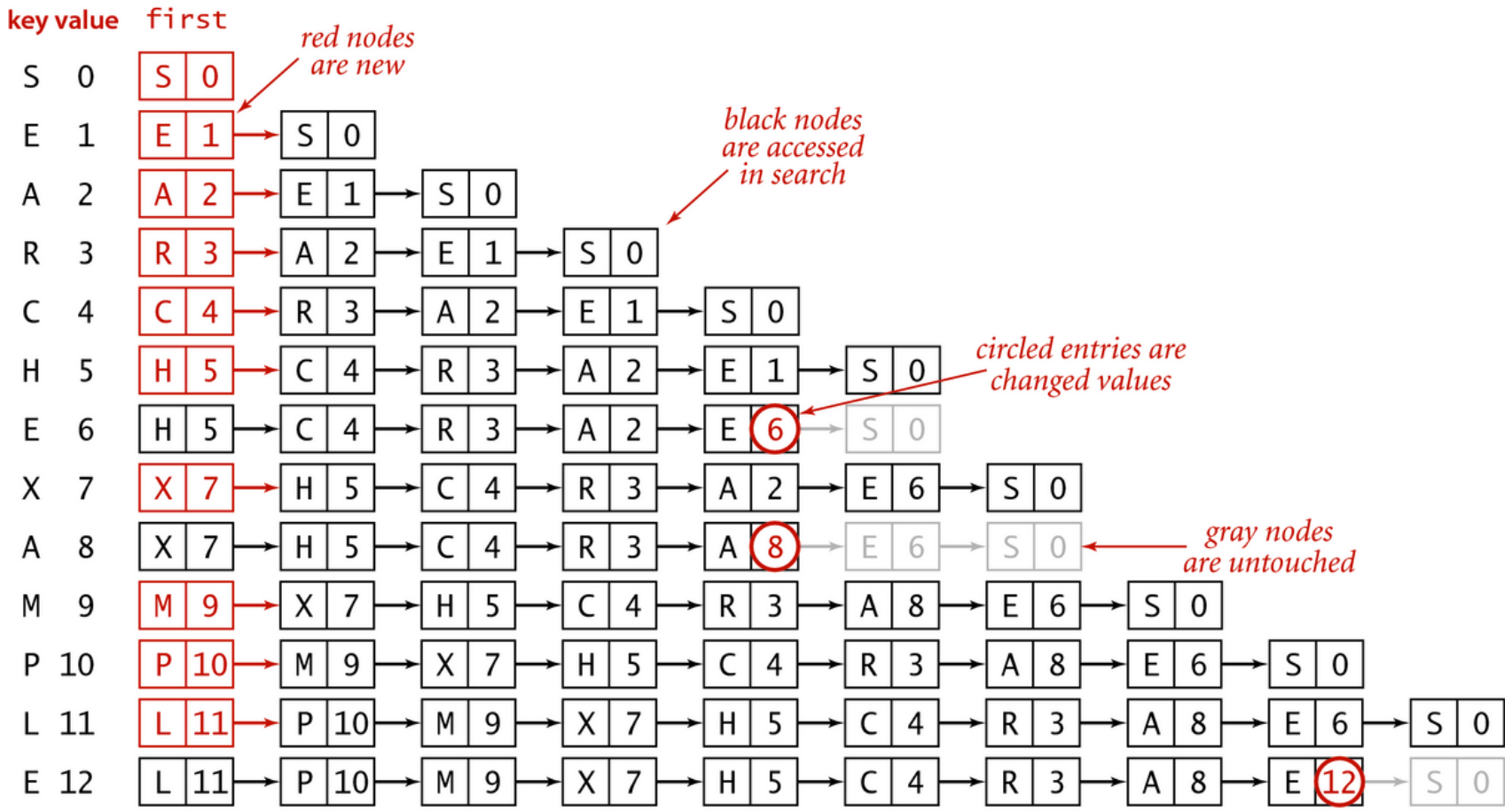
```
public interface SymbolTable<K,V> {
  void put(K key, V value);
  V get(K key);
  int size();
  Iterable<K> keys();
  default void delete(K key) { put(key, null); }
  default boolean contains(K key) {
    return get(key) != null;
    }
  default boolean isEmpty() { return size() == 0; }
  }
```

Ordered Symbol Tables

`OrderedMap` in Java

```
public interface
    OrderedSymbolTable<K extends Comparable<K>, V>
    extends SymbolTable<K,V> {
  K min();
  K max();
  K floor(K key);
  K ceiling(K key);
  int rank(K key);
  K select(int rank);
  void deleteMin();
  void deleteMax();
  int size(K low, K high);
  Iterable<K> keys(K low, K high);
  }
```

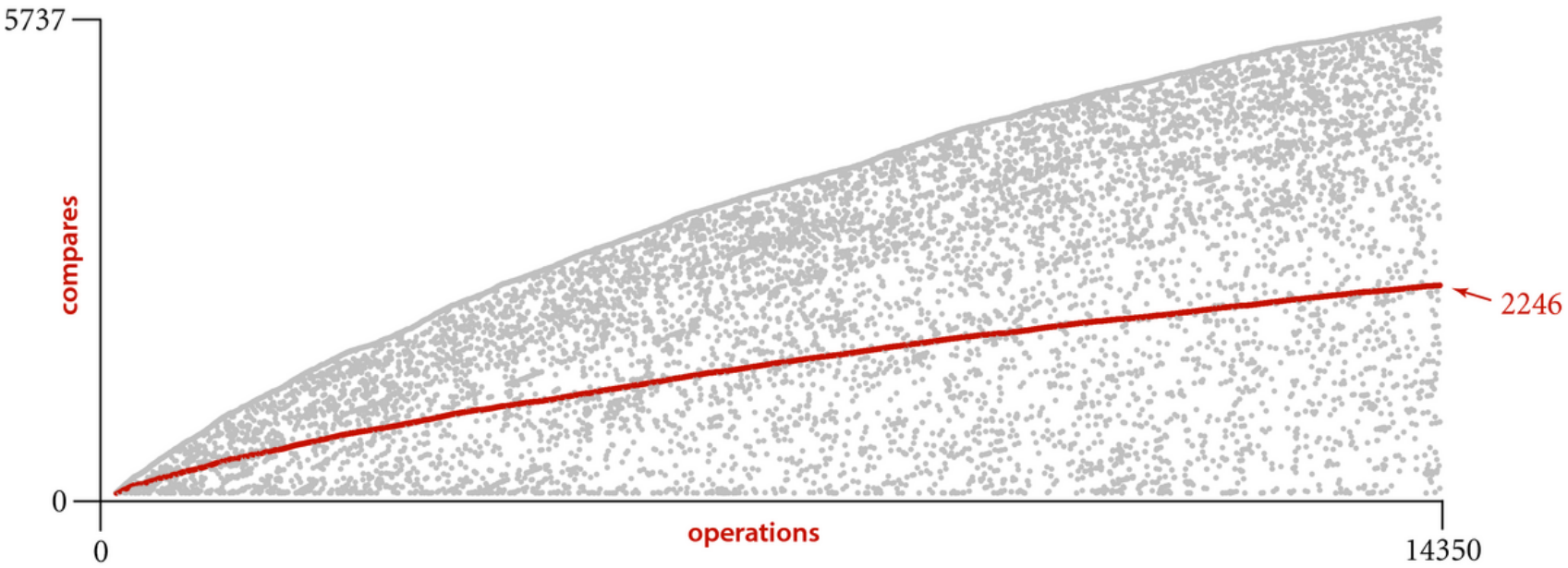
List-based Symbol Tables



Trace of linked-list ST implementation for standard indexing client

cphbusiness
COPENHAGEN BUSINESS ACADEMY

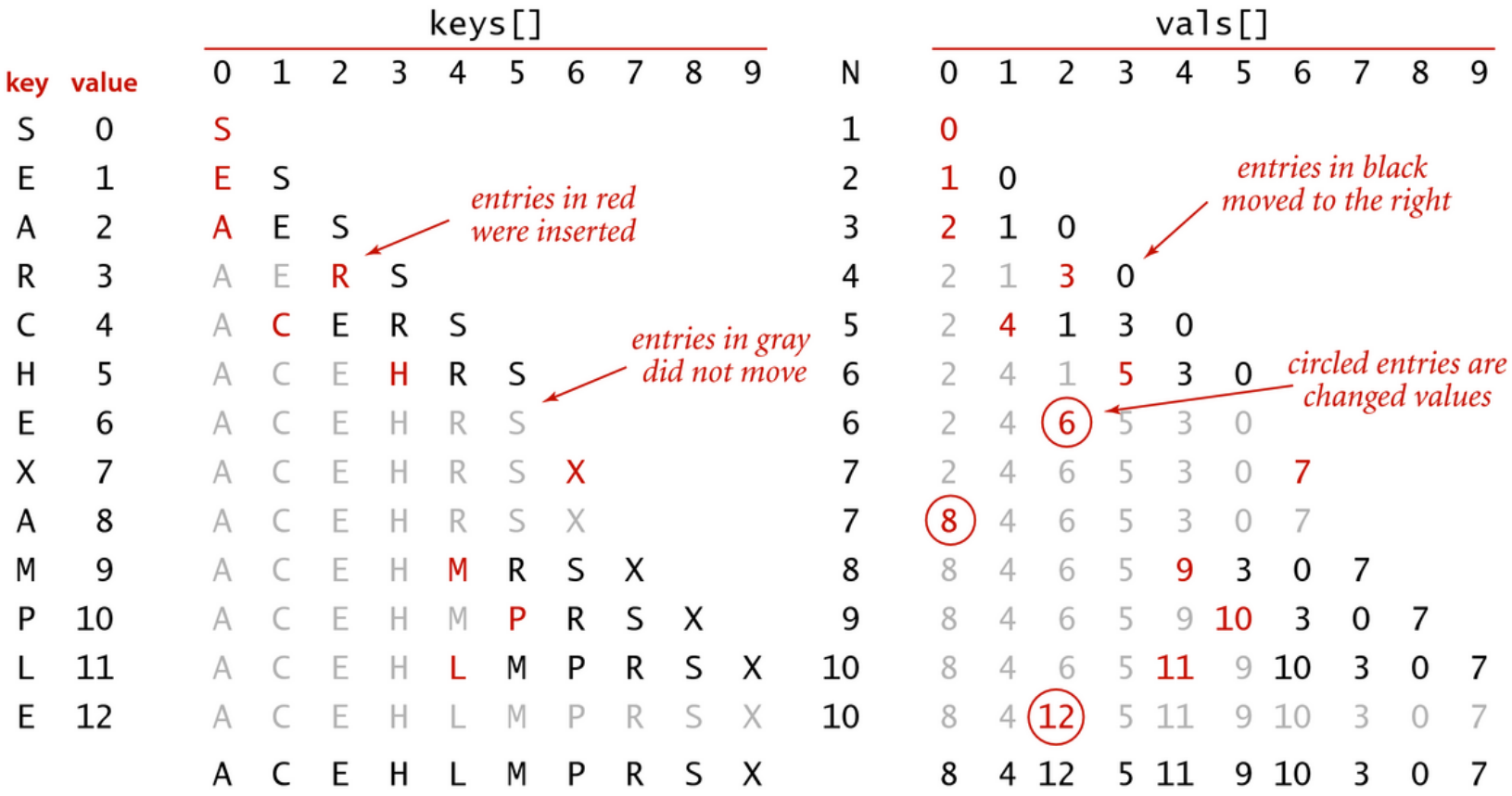
Costs



Costs for `java FrequencyCounter 8 < tale.txt` **using** `SequentialSearchST`

Q: go to the book (Ch.3.1, subsect. 'Performance Client') and figure out what the input argument '8' does here? Discuss in Plenum.

Array-based Symbol Tables

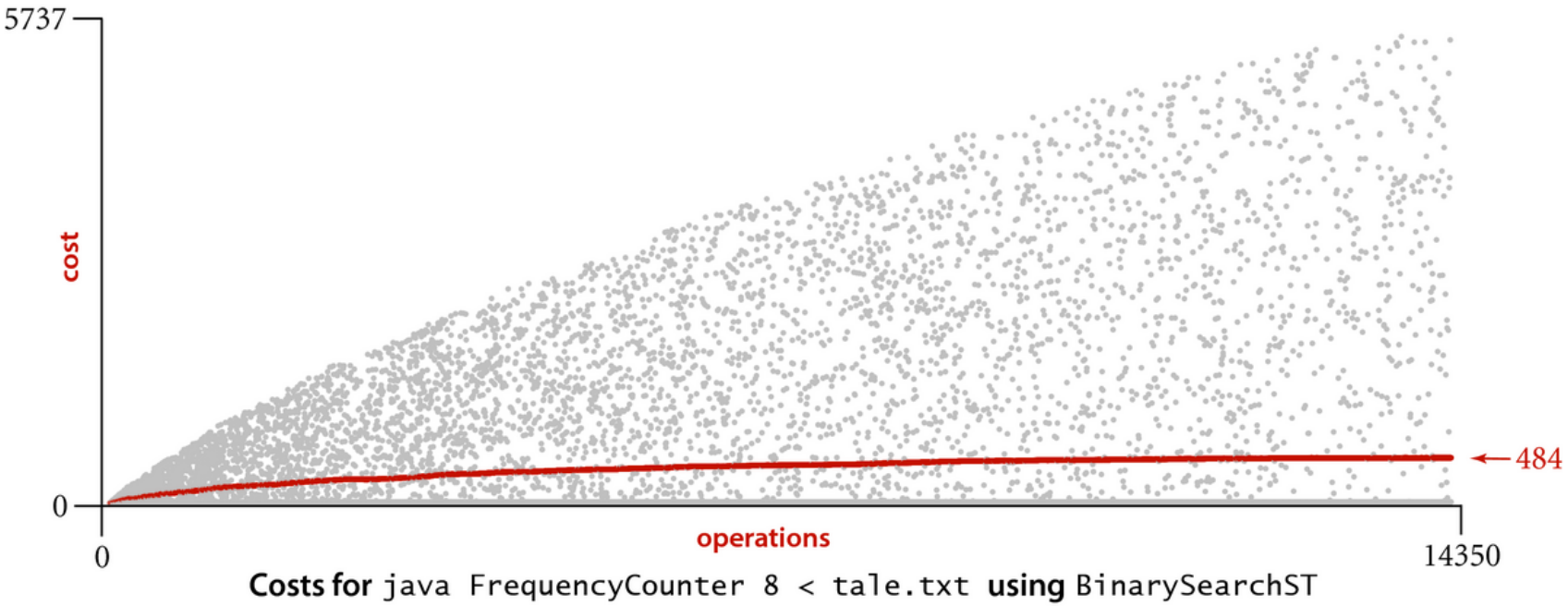


key	value	N	keys[] 0	1	2	3	4	5	6	7	8	9	vals[] 0	1	2	3	4	5	6	7	8	9
S	0	1	S										0									
E	1	2	E	S									1	0								
A	2	3	A	E	S								2	1	0							
R	3	4	A	E	R	S							2	1	3	0						
C	4	5	A	C	E	R	S						2	4	1	3	0					
H	5	6	A	C	E	H	R	S					2	4	1	5	3	0				
E	6	6	A	C	E	H	R	S					2	4	6	5	3	0				
X	7	7	A	C	E	H	R	S	X				2	4	6	5	3	0	7			
A	8	7	A	C	E	H	R	S	X				8	4	6	5	3	0	7			
M	9	8	A	C	E	H	M	R	S	X			8	4	6	5	9	3	0	7		
P	10	9	A	C	E	H	M	P	R	S	X		8	4	6	5	9	10	3	0	7	
L	11	10	A	C	E	H	L	M	P	R	S	X	8	4	6	5	11	9	10	3	0	7
E	12	10	A	C	E	H	L	M	P	R	S	X	8	4	12	5	11	9	10	3	0	7
			A	C	E	H	L	M	P	R	S	X	8	4	12	5	11	9	10	3	0	7

Trace of ordered-array ST implementation for standard indexing client

Array-based Symbol Tables

Costs



Q: What happened here? Look at that decrease in cost... Discuss in plenum.

Costs Comparison

Unordered sequential search *vs.* Ordered array binary search.

algorithm (data structure)	worst-case cost (after N inserts)		average-case cost (after N random inserts)		efficiently support ordered operations?
	search	insert	search hit	insert	
sequential search (unordered linked list)	N	N	$N/2$	N	no
binary search (ordered array)	$\lg N$	$2N$	$\lg N$	N	yes

Cost summary for basic symbol-table implementations

Q: What about insert (or `put()`)? What is *its* complexity? And can we achieve better performance? Discuss in plenum, 'tales.txt' and '.

Array-based Symbol Tables

cphbusiness
COPENHAGEN BUSINESS ACADEMY

Pros & Cons in Searching

underlying data structure	implementation	pros	cons
linked list (sequential search)	`SequentialSearchST`	best for tiny STs	slow for large STs
ordered array (binary search)	`BinarySearchST`	optimal search and space, order-based ops	slow insert
binary search tree	`BST`	easy to implement, order-based ops	no guarantees space for links
balanced BST	`RedBlackBST`	optimal search and insert, order-based ops	space for links
hash table	`SeparateChainingHashST` `LinearProbingHashST`	fast search/insert for common types of data	need hash for each type no order-based ops space for links/empty

Pros and cons of symbol-table implementations

Q: how can we get better e.g. `put()` performance? Need more than linked lists and binary searches=> Datastructures.

Exercise 3 - Sequential Search

NB: These last exercises are optional, but will definitely help you to understand the linked lists, search complexity, and timing issues better.

- Add to the `Algorithm 3.1` in the Book (seq. search in unordered list), the methods mentioned in the text, namely
  - `size()`,
  - `keys()`,

  so essentially exercise 3.1.5, yet, without the *eager* `delete()`.
- Exercise 3.1.6 of the Book.
- What's frequent most word of 10+ letters in *'Tale of two cities'* (ex. 3.1.8 of the book)?