

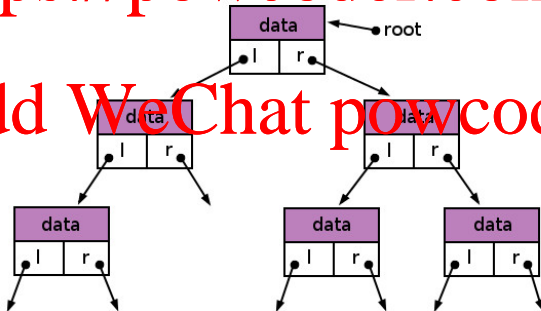
Dr Timothy Kimber

## Assignment Project Exam Help

January 2018

<https://powcoder.com>

Add WeChat powcoder



# Dynamic Data Structures

Having efficient data structures is crucial for successful algorithms.

# Assignment Project Exam Help

- The problems seen so far involved fixed length lists
- In most languages we have a simple way to implement this efficiently — arrays
- Our algorithms assumed some sort of array type was available

Other problems require dynamic data structures such as

- Lists
- Stacks and Queues
- Sets and Dictionaries

These are designed to hold variable, essentially unlimited amounts of data.

<https://powcoder.com>

Add WeChat powcoder

# Ordered Data Structures

A *list* is an ordered collection of {nodes, items, elements}.

- The key property of a list is the ordering of the nodes
- A list might support operations such as

`push` adds an element to the end of the list

`pop` removes the last element of the list

`shift` removes the first element of the list

`unshift` adds an element to the front of the list

`insert` adds an element at a given position

`remove` removes the element at a given position

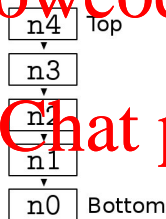
`iterate` returns the items in order

- Plus sorting, searching, copying, joining, splitting ...
- The most appropriate implementation depends on which operations are needed.

# Stacks

A *stack* is a *last-in first-out* (LIFO) list.

- Stacks support only
  - push* for adding elements
  - pop* for removing elements
- Stacks are usually pictured as a vertical (stacked!) structure



- Stacks support recursive algorithms including fundamental operations such as calling subprocedures and evaluating arithmetic expressions

## Stacks

# Assignment Project Exam Help

## Question

How would you implement a stack?

- Must be able to add “unlimited” objects
- Push and Pop must implement LIFO behaviour

<https://powcoder.com>  
Add WeChat powcoder

## Performance of Push

### Question

Given a stack containing  $N$  objects, what is the worst case time complexity of `push`?

- Assume: time to insert (copy, add) one object to array is  $c$
- Assume: initial capacity is 4

<https://powcoder.com>

Add WeChat powcoder

## Performance of Push

### Revised Question

Given an empty stack, what is the worst case time to push  $N$  objects?

- Assume: initial capacity is 4
- Assume: time to insert (copy, add) one object to array is  $c$

<https://powcoder.com>

Add WeChat powcoder

# Amortisation

The time for  $N$  pushes is  $\mathcal{O}(N)$ , so:

- A single push is *effectively* a constant time operation
- More correctly: push is **amortised**  $\Theta(1)$
- NOT the same as  $\Theta(1)$

<https://powcoder.com>

## Amortisation

- Related to accountancy method used to defer large costs
- **Amortised analysis** considers a sequence of operations
- Cost of individual ops is “amortised” across the sequence
- Unlike accountancy, must never be in debt

Add WeChat powcoder



## Amortised Analysis

# Assignment Project Exam Help

Rather than calculating cost of full sequence of  $V$  steps we can

- Pick a **representative** subsequence
- Subsequence is some “cycle” that repeats
- Pick an amortised cost for operations
- Show that paying amortised cost covers all costs (never in debt)

<https://powcoder.com>

### Exercise

Add WeChat powcoder

Find a representative cycle (subsequence) of pushes into the stack and show that the amortised cost of  $3c$  covers all costs.

## Amortised Analysis

Begin cycle when ...  

# Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

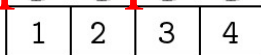
End cycle when ...

# Amortised Analysis

Argument only works because array is initially empty and size is **doubled**

- Say we have  $N$  objects on stack after a copy
- Before next copy we always push  $N$  more
- This is how cost is covered

<https://powcoder.com>



Add WeChat powcoder

A diagram showing an empty array with eight cells, representing the state after the first copy operation.

- Multiplying by **any factor** will do - will affect amortisation constant

## Queues

# Assignment Project Exam Help

7. A queue is also a list, but the next object removed is either

- The **earliest** one added (FIFO Queue)
- The one with highest **priority** (Priority Queue)

<https://powcoder.com>

## Questions

- How could you implement a priority queue (PQ)?
- Given a PQ following your design that contains  $N$  objects, what would be the worst case time to add a new object? (Each object has a key attribute that determines its priority.)

Add WeChat powcoder

## Priority Queue Design

# Assignment Project Exam Help

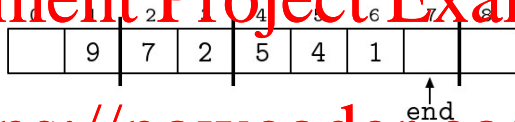
<https://powcoder.com>

Add WeChat powcoder

## Heap: a Tree in an Array

We want to know where the “end” of the tree is:

- Build a tree within an array



<https://powcoder.com>

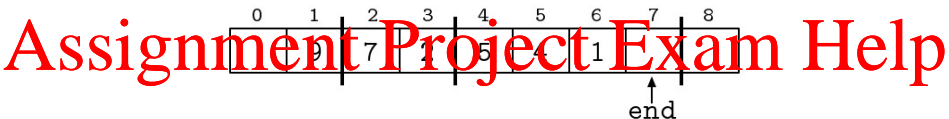
- Track end using “stack pointer”
- Navigate by indices
- Leaving a [0] blank means:

- parent of  $a[n]$  is  $a[n/2]$
- children of  $a[n]$  are  $a[2*n]$  and  $a[2*n+1]$

### Exercise

How should a new object be added to a **max** binary heap? (i.e. the greatest key should be at the root).

## Heap: a Tree in an Array



- <https://powcoder.com>
- Track end using “stack pointer”
  - Leaving  $a[0]$  blank means:

- parent of  $a[i]$  is  $a[\lfloor i/2 \rfloor]$
  - children of  $a[i]$  are  $a[2*i]$  and  $a[2*i+1]$
- Add WeChat powcoder

### Exercise

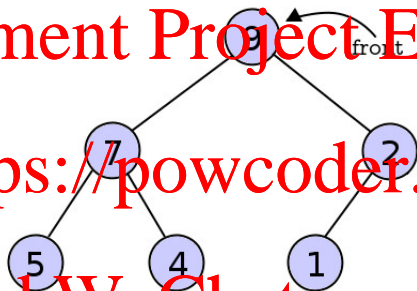
How should the object with the greatest key be removed from a **max** binary heap?

# Binary Heap Performance

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



## Question

Given a heap containing  $N$  objects, what is the time complexity for adding or removing one object?



# Heapsort

Heaps also provide us with the **Heapsort** algorithm (JWJ Williams, 1964)

Heapsort (given a list  $L$ )

- Create an empty heap  $H$
- Remove each element of  $L$  and add it to  $H$
- Remove each element of  $H$  and add it to  $L$
- HALT

- What could be simpler?
- Performance is again  $\Theta(N \log_2 N)$
- Can also be implemented **in place** by setting up list and heap partitions within a single array

# Sets

## Assignment Project Exam Help

7. Set is an unordered collection of objects each having a **unique** key.

- Should have “unlimited” capacity
- Want to **put** and **get** by key
- A key could be any type that defines  $<$  and  $=$

<https://powcoder.com>

### Questions

- How could you implement a set?
- Given a set following your design that contains  $N$  objects, what would be the worst case time to get the object with key  $k$ ?

Add WeChat powcoder

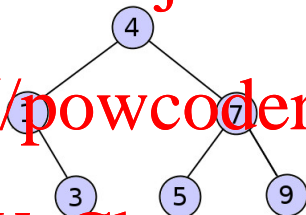
## A Search Tree?

A tree will divide the data but need a different ordering

# Assignment Project Exam Help

<https://powcoder.com>

## Add WeChat powcoder



- Start at the root (it's a tree)
- Go right: find/add larger keys
- Go left: find/add smaller keys

# Binary Search Tree

In a Binary Search Tree

- Go right: find/add larger keys
- Go left: find/add smaller keys

Exercise

<https://powcoder.com>

- Draw the (integer) binary search tree implied by the following code:

```
bst = new BST
keys = [5, 3, 10, 1, 6, 9, 8, 0, 4]
for i = 0 to 8
    bst.put(keys[i])
```

- What is the worst case time complexity of the put procedure?

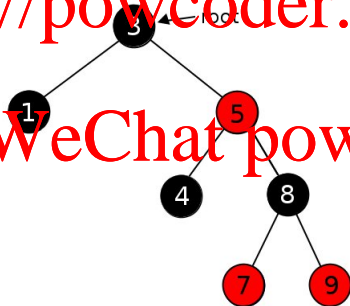
# Red-Black Trees

Red-Black Trees are binary search trees that maintain balance

- A BST can become (very) unbalanced, resulting in long branches
- Searching a BST takes  $O(N)$  time in the worst case
- The branches of a balanced tree remain as short as possible

<https://powcoder.com>

Add WeChat powcoder



## Red-Black Tree Properties

# Assignment Project Exam Help

Definition (Red-Black Tree)

A binary search tree  $T$  is a **red-black tree** iff  $T$  satisfies the following five properties:

- 1 All nodes (including null) are either red or black.
- 2 The root node is black.
- 3 Every leaf (all null) is black.
- 4 Both children of a red node are black.
- 5 All paths from a node to a descendant leaf contain the same number of black nodes.

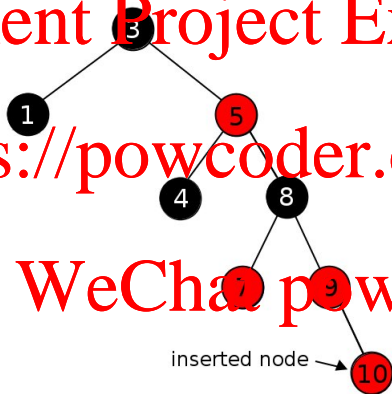
# Insertion

A node is inserted using the ordinary BST procedure

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



- A new node is always colored red

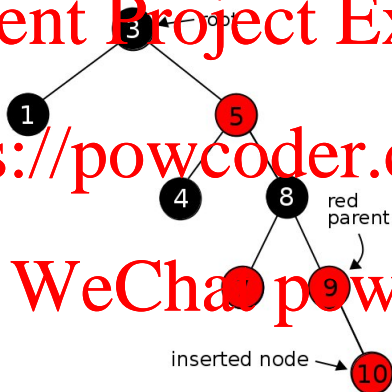
# Insertion

The insertion may result in a violation of the red-black tree properties

# Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



- The root might be coloured red
- A red node might have a red child



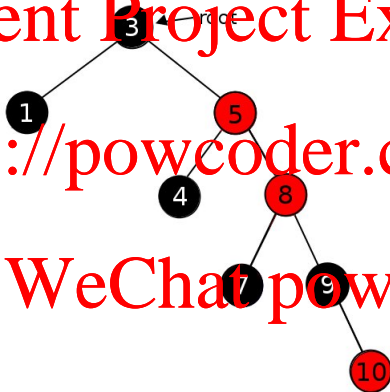
# Insertion

Either recolour  $\Theta(1)$  nodes

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



- There is still a red node with a red parent
- The problem has moved closer to the root (continue)

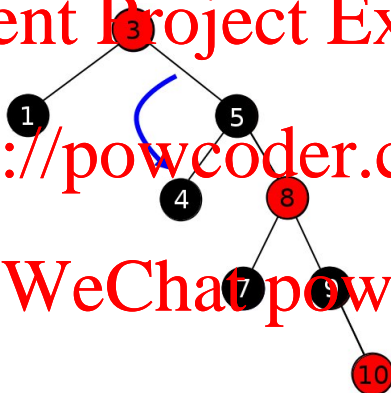
# Insertion

Or perform a **rotation** of  $\Theta(1)$  nodes and **Stop**

# Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



- Reduces height of the tree
- Preserves key ordering

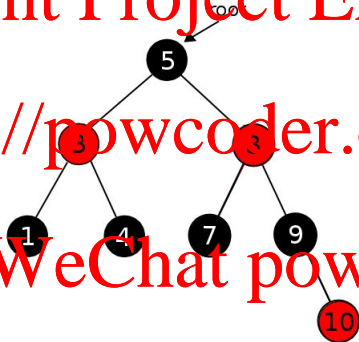
# Insertion

The properties are restored

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



## Performance

# Assignment Project Exam Help

By maintaining the red-black tree properties, we have  $h \leq 2\log_2(N + 1)$

- Get procedure is the same as for BST
- Height constraint means it is now  $O(\log_2 N)$

For Put, only the last part is different

- The extra work is still localised to one branch
- So, Put also runs in  $O(\log_2 N)$  time

<https://powcoder.com>  
Add WeChat powcoder