

Dr Timothy Kimber

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

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 m
 - at
- Our

<https://eduassistpro.github.io>

Other problems require *dynamic* data structures such

- Lists, Stacks and Queues
- Sets and Dictionaries

Add WeChat edu_assist_pro

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

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

<https://eduassistpro.github.io>

`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

- Sta

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

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

Stacks

Assignment Project Exam Help

Question

How would

<https://eduassistpro.github.io>

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

Add WeChat edu_assist_pr

Stack Implementation

Could use array or linked-list as storage model

- Both very simple implementation
- Arr
- Lin

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pro

- Can make array **dynamic** (variable size)
- Integer `sp` points to the top of the stack
- Update `sp` within Push and Pop

Dynamic Array-Based Stack

Assignment Project Exam Help

```

push(input: stack s[0, ..., k-1], object x)
    if s.sp == k                                // array is full
        s = s'
    s[s.sp] = x
    s.sp = s.sp + 1

```

- push increases the capacity of the stack if it is full

Dynamic Array-Based Stack

```
Pop(input: stack s[0..k-1])  
if s.sp < k/2 // array is less than half full
```

```
    s
```

```
    f
```

```
    s.sp = s.sp - 1  
    return s[s.sp]
```

- pop decreases the capacity if it is too big
- a full implementation should have a minimum size

Performance of Push

Question

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

- Ass

c

- Ass

The time taken for pushing objects is:

- $c, c, c, c, c + 4c, c, \dots$

So:

- Worst time to push a single object is Nc

- So $T(N) = O(N)$

- Want to reflect fact that most pushes are not $O(N)$

Performance of Push

Assignment Project Exam Help

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

- Ass
- Ass

<https://eduassistpro.github.io>

The time taken for the each push is still:

- $c, c, \dots, c, c = 4c, c, \dots$
- For N pushes the worst single push is
- $T(N) = O(N^2)$

Add WeChat edu_assist_pro

However, this is a big overestimate

Performance of Push

The time for N pushes is at most

$$T(N) = Nc + (4c + 8c + \dots + (N/2)c + Nc)$$

where

- the first $4c$ is for the first push
- the rest is for copying to new arrays

The time for copying is $2Nc - 4c$ (see note)

- $T(N) \leq 3Nc - 4c$
- $T(N) = O(N)$

Amortisation

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

- A single push is *effectively* a constant time operation
- Mo
- NO

Amortis

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

Amortised Analysis

Assignment Project Exam Help

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

- Pick a **representative** subsequence
- Sub
- Pic
- Show that paying amortised cost covers all costs (never in debt)

<https://eduassistpro.github.io>

Exercise

Add WeChat edu_assist_pro

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

Amortised Analysis

- Start **after** any expensive push
- Up to and including next expensive one

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Cheap pushes each put $2c$ in the bank
- Have enough to cover extra costs when next expensive push occurs
- (If you started with a copy you are immediately over budget)

Amortised Analysis

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

Assignment Project Exam Help

- Say we have N objects on stack after a copy
- Before next copy we always push N more

- Thi

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

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

Queues

Assignment Project Exam Help

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

- The **earliest** one added (FIFO Queue)
- The

<https://eduassistpro.github.io>

Question

- How could you implement a priority queue (PQ)
- Given a PQ following your design that contains *Add WeChat edu_assist_pr* would be the worst case time to add a new object? (E a key attribute that determines its priority.)

Priority Queue Design

If we maintain a total ordering of the queue:

- It will take $O(N)$ time to add new elements
- Can search a sorted array quickly but have to shift existing objects
- Fin

<https://eduassistpro.github.io>

9, 8, 7, 6, 3
remove items

Max Priority Queue

6, 9, 7, 3, 8

Add WeChat edu_assist_pr

Do not actually need total ordering.

- Queue does not support indexed access
- Just want to find object with highest priority

Priority Queue Design

Solution is to divide and conquer the data

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pro

- **Key Property:** Maintain order within each branch
- Highest (or lowest) key will be at the root
- Behaves like lots of mini queues

Priority Queue Design

Solution is to divide and conquer the data

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pro

Question

A new object could go in any branch. (Do you agree?) So, where should it go? Why?

Heap: a Tree in an Array

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

- Build a tree within an array

Assignment Project Exam Help

<https://eduassistpro.github.io>

- Tra
- 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 $a[2*n+1]$

Add WeChat edu_assist_pro

Exercise

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

Binary Heap Operations

Adding an object to the heap

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Operations

Adding an object to the heap

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Operations

Adding an object to the heap

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Operations

Adding an object to the heap

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Heap: a Tree in an Array

Assignment Project Exam Help

- Tra <https://eduassistpro.github.io>
- Leaving $a[0]$ blank means:
 - parent of $a[i]$ is $a[i/2]$
 - children of $a[i]$ are $a[2*i]$ and $a[2*i+1]$

Exercise

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

Binary Heap Operations

To remove the object at the root

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Operations

To remove the object at the root

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Operations

To remove the object at the root

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Operations

To remove the object at the root

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Operations

To remove the object at the root

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pr

- Restore the “shape”
- Then restore the order

Binary Heap Performance

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pro

Question

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

Binary Heap Performance

Assignment Project Exam Help

<https://eduassistpro.github.io>

Add WeChat edu_assist_pro

Both operations are $O(\log_2 N)$

- Height of the heap is $\Theta(\log_2 N)$
- Each operation confined to one branch

Heapsort

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

Heapsort (given a list L)

- Cre
- Re
- Re
- 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