# Final Exam

## Thu 19th August 2021

## Changelog

All changes to the exam paper and files will be listed here.

-

## Rules & Overview

### General

- By completing the acknowledgement and starting this exam you have acknowledged that you are fit to sit the exam and cannot apply for Special Consideration for issues that existed prior to the exam
- If during the exam a circumstance arises that prevents you from completing the exam, please email cs2521@cse.unsw.edu.au immediately and apply for special consideration shortly after
- During the exam no communication is allowed with other people, excluding any common sense exceptions (e.g. answering "how are you?" if a parent asks you)
- If you have any questions during the exam, make a **private** post on the Ed forum (the same one we've been using all term)
- **Do not wait until just before the deadline to submit all your answers. Submit each question as soon as you finish working on it or submit incrementally throughout the exam.**

### Programming Questions

- All inputs will be valid.
- You may define your own helper functions.
- You **may not** `#include` any additional libraries.
- You may add your ns if you require them for your solution.
- You may use globa
- Solutions which d out instead only hardcode return values for specific tests will receive zero marks

## Admin

rk

**S** h question

**Date an**

**Total**

**Total number of questions** 14 (not worth equal marks)

## Structure

This exam consists of a series of questions:

- 30 marks for written short/extended answers
- 70 marks for programming questions

# Setting Up

Change into the directory you created for the sample exam and run the following command:

```
$ unzip /web/cs2521/21T2/final-exam/downloads/files.zip
```

If you're working at home, download `files.zip` by clicking on the above link and then unzip the downloaded file.

# Question 1 (4 marks)

Consider the following scenario:

```
I need a graph with 10,000 vertices but it will only ever have 10-20 edges in the graph at any given time. The
priority needs to be space efficiency above all else - it doesn't matter how slow operations are, as long as
it's using the least possible memory.
```

Of the 3 graph representations discussed in the course, which would be most appropriate for use in this scenario? Justify your answer.

Write your answer in `q1.txt`

**Expected response size: 30-100 words**

## Submission

Submit via the give command

```
$ give cs2521 exam_q1 q1.txt
```

# Question 2 (4 marks)

Consider the following scenario:

```
I'd like to produce a hash table which has a key space of 10,000 and an index space of 1,000. I intend to add
1,500 keys to the                                       able.
```

Which hash collision                                       rio? Summarise the way this particular method works, and use it to justify your answ

Write your answer in

**Expected response**

## Submission

Submit via the give c

```
$ give cs2521 exa
```

# Question 3

Describe the types of data or scenario(s) in which heap sort would be an ideal sort to use.

Write your answer in `q3.txt`

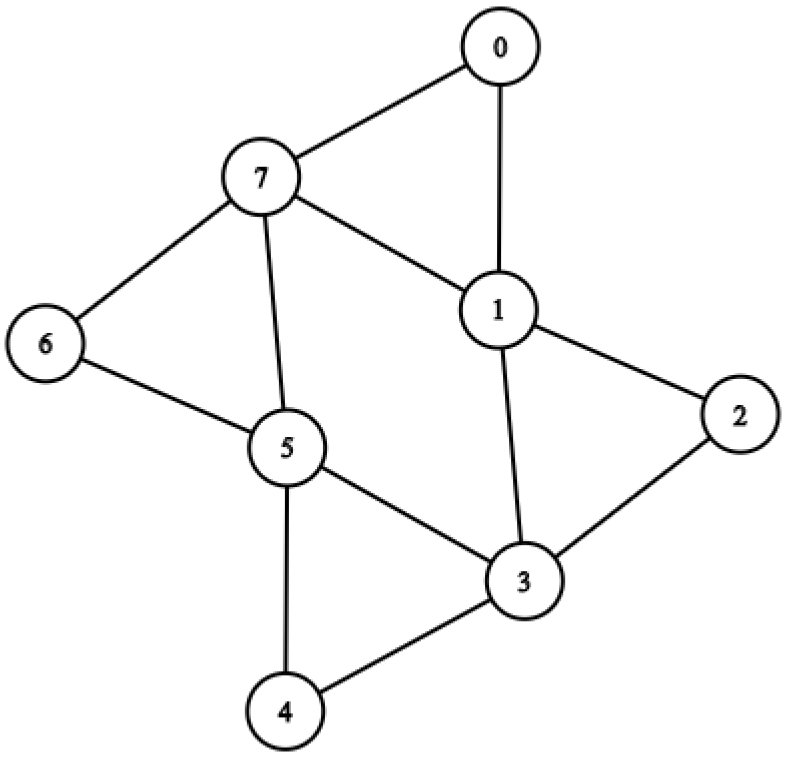
**Expected response size: 30-100 words**

## Submission

Submit via the give command

```
$ give cs2521 exam_q3 q3.txt
```

# Question 4 (3 marks)



Does an Euler path (NOT an Euler circuit) exist for this graph? Justify your answer.

Write your answer in `q4.txt`

**Expected response size: 30-100 words**

## Submission

Submit via the give command

```
$ give cs2521 exam_q4 q4.txt
```

# Question 5 (3 marks)

What is a limitation of recursive solutions (as opposed to iterative solutions)? Describe an example.

Write your answer in [illegible]

**Expected response** [illegible]

## Submission

Submit via the give c[illegible]

```
$ give cs2521 exa[illegible]
```

# Question 6 [illegible]

Consider the followin[illegible]

```
void processThings(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            for (int k = 0; k < 2; k++) {
                printf("%d %d %d\n", i, j, k);
            }
        }
    }
}
```

What is the best and worst case time complexity for the function above?

Write your answer in `q6.txt`

*Provide time complexities with respect to **n**. printf() can be assumed to always be O(1).*

## Submission

Submit via the give command

```
$ give cs2521 exam_q6 q6.txt
```

## Question 7 (3 marks)

Consider the following function:

```
void processThings(int n, int m) {
    int *nums = malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++) {
        nums[i] = randInt(0, n);
    }
    for (int i = 0; i < m; i++) {
        printf("%d\n", i);
    }
    insertionSort(nums, 0, n - 1);
}
```

What is the best and worst case time complexity for the function above?

Write your answer in `q7.txt`

*Provide time complexities with respect to **n** and **m**. Note that **randInt** is a function that generates a random number, and **insertionSort** is a function that uses standard insertion sort to sort an array between two bounds. malloc(), randInt() and printf() can be assumed to always be O(1).*

## Submission

Submit via the give command

```
$ give cs2521 exam_q7 q7.txt
```

## Question 8 (5 marks)

Describe the key diffe

n includes at least:

- Their respective be
- Examples of circur

r the other

*You can assume the*

Write your answer in

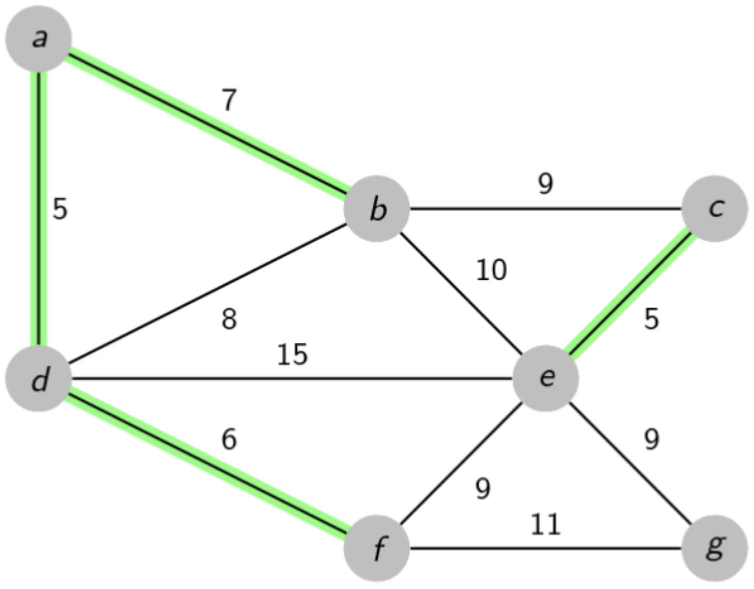
**Expected response**

## Submission

Submit via the give c

```
$ give cs2521 exa
```

## Question 9 (3 marks)



Is this minimum spanning tree being generated with Prim's or Kruskal's algorithm, or could it be generated by either? Justify your answer.

Note: This MST is mid-construction, and the green lines denote what has been constructed so far.

Write your answer in `q9.txt`

**Expected response size: 30-80 words**

## Submission

Submit via the give command

```
$ give cs2521 exam_q9 q9.txt
```

# Question 10 (9 marks)

Implement the following function in the file `q10/equalBST.c`:

```
int equalBST(BSTree t1, BSTree t2);
```

`equalBST` takes two binary search trees. It should return 1 if the trees are equal, and 0 otherwise. Two binary search trees are considered to be equal if they have the same structure and corresponding values.

## Constraints

- The given trees mu
- The time complexi m), where *n* and *m* represent the number of nodes in `t1` and `t2` respectively. If you mark you can attain for this question will be 7 (out of 9). Note that a fairly standa

## Files

| | |
|---|---|
| **BSTree.c** | rch tree functions. |
| **BSTree.h** | data structure and function prototypes. |
| **testEqualBST.c** | ata from `stdin`, calls `equalBST`, and outputs the result to |
| **equalBST.c** | Contains `equalBST`, the function you must implement. |
| **Makefile** | A Makefile to compile your code |
| **tests/** | A directory containing the inputs and expected outputs for some basic test cases |

## Examples

The following are examples of how the program should behave:

```
$ ./testEqualBST
5 3 2 4 7
t1:

    5
   / \
  3   7
 / \
2   4

5 3 2 4 8
t2:

    5
   / \
  3   8
 / \
2   4

equalBST returned: 0
```

```
$ ./testEqualBST
5 3 2 4 7
t1:

    5
   / \
  3   7
 / \
2   4

5 2 4 3 7
t2:

  5
 / \
2   7
 \
  4
 /
3

equalBST returned
```

```
$ ./testEqualBST
5 3 2 4 7
t1:

    5
   / \
  3   7
 / \
2   4

5 3 2 4 7
t2:

    5
   / \
  3   7
 / \
2   4

equalBST returned: 1
```

```
$ ./testEqualBST

t1:



t2:


equalBST returned: 1
```

In the last example, empty trees were created by pressing enter without typing any numbers.

## Testing

You can compile and test your function using the following commands:

```
$ make                               # compiles the program
$ ./testEqualBST                     # tests with manual input, outputs to terminal
$ ./testEqualBST < input-file        # tests with input from a file, outputs to terminal
$ ./testEqualBST < tests/input1.txt  # for example, tests with input from tests/input1.txt
```

## Submission

Submit via the give command

```
$ give cs2521 exam_q10 equalBST.c
```

# Question 11 (14 marks)

Implement the following function in the file `q11/printWords.c`:

```
void printWords(Trie t);
```

`printWords` takes one argument: a pointer to the root node of a Trie. It should print all the words in the trie to stdout in alphabetical order, one per line.

## Assumptions

- Words consist only
- Words are betwee

## Constraints

- The given trie mus

## Files

| | |
|---|---|
| **Trie.c** | associated functions. |
| **Trie.h** | tion prototypes. |
| **testTriePrintWords** | data from `stdin` and calls `printWords`. |
| **printWords.c** | **to modify**. Contains `printWords`, the function you must |
| **Makefile** | A Makefile to compile your code |
| **tests/** | A directory containing the inputs and expected outputs for some basic test cases |

## Examples

The following are examples of how the program should behave:

```
$ ./testTriePrintWords
the they them what is life
 Ctrl-D
is
life
the
them
they
what
```

```
$ ./testTriePrintWords
hello hey
 Ctrl-D
hello
hey
```

## Testing

You can compile and test your function using the following commands:

```
$ make                                      # compiles the program
$ ./testTriePrintWords                      # tests with manual input, outputs to terminal
$ ./testTriePrintWords < input-file         # tests with input from a file, outputs to terminal
$ ./testTriePrintWords < tests/input1.txt   # for example, tests with input from tests/input1.txt
```

## Submission

Submit via the give command

```
$ give cs2521 exam_q11 printWords.c
```

## Question 12 (14 marks)

Implement the following function in the file `q12/mergeOrdered.c`:

```
List mergeOrdered(List list1, List list2);
```

`mergeOrdered` takes two ordered lists, each of which is in non-decreasing order (i.e., increasing order potentially with duplicates). It should merge the t so in non-decreasing order and return the merged list.

## Assumptions

- The given lists will
- The given lists may be added to the merged list in any order.
- You may use the fu

## Constraints

- **The function mus**
- The given lists mus
- You must not use a tisfy this requirement, you will receive zero for this question.

## Files

| | |
|---|---|
| **list.c** | d list functions. |
| **list.h** | Contains the definition of the linked list data structure and function prototypes. |
| **testMergeOrdered.c** | A testing program. The program reads list data from `stdin`, calls `mergeOrdered`, and outputs the result to `stdout`. |
| **mergeOrdered.c** | **This is the only file you're required to modify**. Contains `mergeOrdered`, the function you must implement. |
| **Makefile** | A Makefile to compile your code |
| **tests/** | A directory containing the inputs and expected outputs for some basic test cases |

## Examples

The following are examples of how the program should behave:

```
$ ./testMergeOrdered
1 4 6
list1: 1, 4, 6
2 8 10 15
list2: 2, 8, 10, 15
merged list: 1, 2, 4, 6, 8, 10, 15
```

```
$ ./testMergeOrdered
1 4 6
list1: 1, 4, 6
2 4 4 8 10 20
list2: 2, 4, 4, 8, 10, 20
merged list: 1, 2, 4, 4, 4, 6, 8, 10, 20
```

```
$ ./testMergeOrdered

list1:
7 9 14 55 82
list2: 7, 9, 14, 55, 82
merged list: 7, 9, 14, 55, 82
```

In the last example, an empty list was created by pressing enter without typing any numbers.

## Testing

You can compile and test your function using the following commands:

```
$ make                                      # compiles the program
$ ./testMergeOrdered                        # tests with manual input, outputs to terminal
$ ./testMergeOrdered < input-file           # tests with input from a file, outputs to terminal
$ ./testMergeOrdered < tests/input1.txt     # for example, tests with input from tests/input1.txt
```

## Submission

Submit via the give command

```
$ give cs2521 exam_q12 mergeOrdered.c
```

## Question 13 (14 marks)

Implement the followi d.c.

```
int nodesNotBalan
```

nodesNotBalance It should return the number of nodes in the tree that are **not** height-balanced.

## Constraints

- The given tree mus
- You must **not** use r indirectly. If you don't satisfy this requirement, you will receive zero for this questi

## Files

| | |
|---|---|
| **BSTree.c** | Contains the implementation of basic binary search tree functions. |
| **BSTree.h** | Contains the definition of the binary search tree data structure and function prototypes. |
| **testNodesNotBalanced.c** | A testing program. The program reads tree data from `stdin`, calls `nodesNotBalanced`, and outputs the result to `stdout`. |
| **nodesNotBalanced.c** | **This is the only file you're required to modify**. Contains `nodesNotBalanced`, the function you must implement. |
| **Makefile** | A Makefile to compile your code |
| **tests/** | A directory containing the inputs and expected outputs for some basic test cases |

## Examples

The following are examples of how the program should behave:

```
$ ./testNodesNotBalanced
5 6
Tree:

5
 \
  6

nodesNotBalanced returned: 0
```

```
$ ./testNodesNotBalanced
5 6 7
Tree:

5
 \
  6
   \
    7

nodesNotBalanced returned: 1
```

```
$ ./testNodesNotBalanced
5 3 2 1 4 7 6 8 9
Tree:

        5
       / \
      /   \
     /     \
    3       7
   / \     / \
  2   4   6   8
 /             \
1               9

nodesNotBalanced returned: 0
```

```
$ ./testNodesNotBalanced
5 4 3 2 1 7 6 8
Tree:

        5
       / \
      4   7
     /   / \
    3   6   8
   /
  2
 /
1

nodesNotBalanced
```

## Testing

You can compile and s:

```
$ make                                         # compiles the program
$ ./testNodesNotBalanced                       # tests with manual input, outputs to terminal
$ ./testNodesNotBalanced < input-file          # tests with input from a file, outputs to terminal
$ ./testNodesNotBalanced < tests/input1.txt    # for example, tests with input from tests/input1.txt
```

## Submission

Submit via the give command

```
$ give cs2521 exam_q13 nodesNotBalanced.c
```

## Question 14 (19 marks)

Implement the following function in the file `q14/rankPopularity.c`:

```
void rankPopularity(Graph g, int src, double *mnV);
```

`rankPopularity` takes three arguments: a directed graph `g`, a source node `src`, and an array `mnV`. For each node *reachable* from `src` (*and only these nodes*), the function should calculate the popularity rank of that node and store it in the array `mnV`. For example, `mnV[2]` should contain the popularity rank of node 2 if node 2 is reachable from `src`.

> **Popularity Rank**
>
> We can calculate the popularity rank of a node `v` using the following formula:
>
> ```
> popularityRank(v) = (inDegree(v) / outDegree(v))
> ```
>
> If `outDegree(v)` is zero, use 0.5 as the denominator instead to avoid division by zero.
>
> Please read the example below for more clarifications.

**Important:** If a node is not reachable from `src`, the function should not calculate and store the popularity rank of that node. Only nodes reachable from `src` should be considered. In the test program `testRankPopularity.c`, all values in `mnV` are initialised to -1.0. If a node `v` is not reachable, `mnV[v]` should remain as -1.0. If you change the value, you will fail the test.

## Assumptions

- You can assume that the array `mnV` has size `n`, where `n` is the number of nodes in the graph.

## Constraints

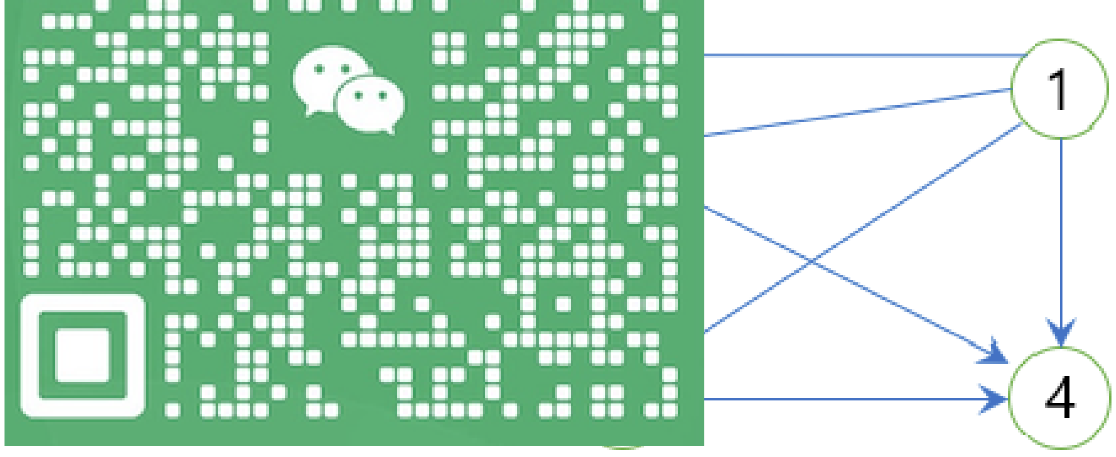
- The given graph must not be modified.

## Files

| | |
|---|---|
| **Graph.c** | Contains the implementation of basic Graph ADT functions. |
| **Graph.h** | Contains the definition of the Graph ADT and function prototypes. |
| **testRankPopularity.c** | A testing program. The program reads graph data, calls `rankPopularity`, and outputs the results to `stdout`. |
| **rankPopularity.c** | **This is the only file you're required to modify**. Contains `rankPopularity`, the function you must implement. |
| **Makefile** | A Makefile to compile your code |
| **tests/** | cted outputs for some basic test cases |

## Example

For example, conside



The popularity ranks for nodes reachable from node 3 in the above graph are:

- popularityRank(0) = 3/1 = 3.0
- popularityRank(2) = 2/2 = 1.0
- popularityRank(3) = 1/2 = 0.5
- popularityRank(4) = 3/0.5 = 6.0 (since outDegree(4) = 0, we have used 0.5 instead)

Note that node 1 is not reachable from node 3, so it must be ignored and its popularity rank must remain as -1.

## Testing

You can compile and test your function using the following commands:

```
$ make                                      # compiles the program
$ ./testRankPopularity < input-file         # tests with input from a file, outputs to terminal
$ ./testRankPopularity < tests/input1.txt   # for example, tests with input from tests/input1.txt
```

## Submission

Submit via the give command

```
$ give cs2521 exam_q14 rankPopularity.c
```

This is the end of the exam.

**COMP2521 21T2: Data Structures and Algorithms** is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2521@cse.unsw.edu.au
CRICOS Provider 00098G