TCP2101 Algorithm Design and Analysis **Khor Kia Kin**
Assignment - Question 4 **1142701883**

Instruction

To compile: **g++ -std=c++11 PriorityQueue.cpp main.cpp**

Or run **build.bat** for Windows user.

Alternatively, run **FractionalKnapsack.cbp** using CodeBlocks.

Screenshots

1) Test - Manual Input

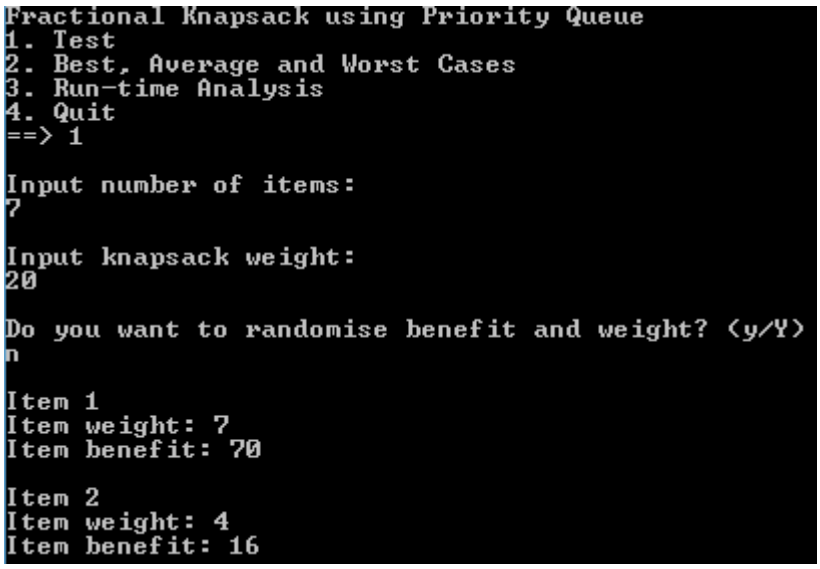
User has to input

- Number of items,
- Knapsack weight,
- Weight and benefit of every item.

The output will be

- Priority queue – item number, benefit, weight, and value.
- Fractional knapsack – item number, portion, benefit, weight, value, total weight so far, and total value so far.
- Total value and weight from fractional knapsack algorithm.
- Total time taken for the algorithm.

The example is taken from lab 8.



```
Item 3
Item weight: 3
Item benefit: 45

Item 4
Item weight: 9
Item benefit: 45

Item 5
Item weight: 8
Item benefit: 40

Item 6
Item weight: 4
Item benefit: 80

Item 7
Item weight: 5
Item benefit: 10


Priority Queue:
No     | Benefit | Weight | Value
    6 |      80 |      4 | 20.00
    3 |      45 |      3 | 15.00
    1 |      70 |      7 | 10.00
    4 |      45 |      9 |  5.00
    2 |      16 |      4 |  4.00
    5 |      40 |      8 |  5.00
    7 |      10 |      5 |  2.00


Fractional Knapsack:
No     | Portion | Benefit | Weight | Value | Total Weight | Total Value
    6 | 100.00% |      80 |      4 | 20.00 |            4 |       80.00
    3 | 100.00% |      45 |      3 | 15.00 |            7 |      125.00
    1 | 100.00% |      70 |      7 | 10.00 |           14 |      195.00
    5 |   0.75% |      40 |      8 |  5.00 |           20 |      225.00

Total weight: 20
Total value: 225.00

Duration: 0.031250s
```

2) Test – Randomise Weight and Benefit

User has to input

- Number of items,
- Knapsack weight,
- Max item weight and item benefit for the randomisation.

Output will be

- Priority queue – item number, benefit, weight, and value.
- Fractional knapsack – item number, portion, benefit, weight, value, total weight so far, and total value so far.
- Total value and weight from fractional knapsack algorithm.
- Total time taken for the algorithm.

```
Fractional Knapsack using Priority Queue
1. Test
2. Best, Average and Worst Cases
3. Run-time Analysis
4. Quit
==> 1

Input number of items:
10

Input knapsack weight:
25

Do you want to randomise benefit and weight? (y/Y)
y

Max item weight: 50
Max item benefit: 100

Generating items with random weight and benefit...

Priority Queue:
No     | Benefit | Weight | Value
    6 |      75 |      1 | 75.00
    8 |      79 |     14 |  5.64
    1 |      82 |     28 |  2.93
    3 |      17 |      4 |  4.25
    2 |      69 |     26 |  2.65
    5 |      97 |     34 |  2.85
    4 |      13 |     26 |  0.50
    7 |      26 |     35 |  0.74
    9 |      39 |     24 |  1.62
   10 |      27 |     37 |  0.73


Fractional Knapsack:
No     | Portion | Benefit | Weight | Value | Total Weight | Total Value
    6 | 100.00% |      75 |      1 | 75.00 |            1 |       75.00
    8 | 100.00% |      79 |     14 |  5.64 |           15 |      154.00
    3 | 100.00% |      17 |      4 |  4.25 |           19 |      171.00
    1 |   0.21% |      82 |     28 |  2.93 |           25 |      188.57

Total weight: 25
Total value: 188.57

Duration: 0.031245s
```

3) Best, Average and Worst Cases

No input from user is required.

Output will be

- Max item weight and item benefit.
- Duration for different number of items and knapsack weight.

```
Fractional Knapsack using Priority Queue
1. Test
2. Best, Average and Worst Cases
3. Run-time Analysis
4. Quit
==> 2

Max Item Weight: 50000000
Max Item Benefit: 10000000

Number of Items | Knapsack Weight | Duration
         100000 |          100000 | 0.002031s
         100000 |         1000000 | 0.007005s
         100000 |        10000000 | 0.027014s
        1000000 |          100000 | 0.010036s
        1000000 |         1000000 | 0.029022s
        1000000 |        10000000 | 0.105070s
       10000000 |          100000 | 0.047032s
       10000000 |         1000000 | 0.117083s
       10000000 |        10000000 | 0.400309s
```

4) Run-time Analysis

No input from user is required.

Output will be

- Knapsack weight, max item weight and item benefit.
- Duration for different number of items.

```
Fractional Knapsack using Priority Queue
1. Test
2. Best, Average and Worst Cases
3. Run-time Analysis
4. Quit
==> 3

Knapsack Weight: 2000000
Max Item Weight: 50000000
Max Item Benefit: 10000000

Number of Items | Duration
            500 | 0.000000s
           1000 | 0.001000s
           1500 | 0.001001s
          10000 | 0.002001s
          15000 | 0.003008s
         100000 | 0.010007s
         150000 | 0.013010s
        1000000 | 0.046034s
        1500000 | 0.060043s
       10000000 | 0.180127s
       15000000 | 0.240172s
```

Best, Average and Worst Cases

Max Item Weight: 50,000,000

Max Item Benefit: 10,000,000

Number of Items	Knapsack Weight	Duration (s)
100,000	100,000	0.002031
100,000	1,000,000	0.007005
100,000	10,000,000	0.027014
1,000,000	100,000	0.010036
1,000,000	1,000,000	0.029022
1,000,000	10,000,000	0.105070
10,000,000	100,000	0.047032
10,000,000	1,000,000	0.117083
10,000,000	10,000,000	0.400309

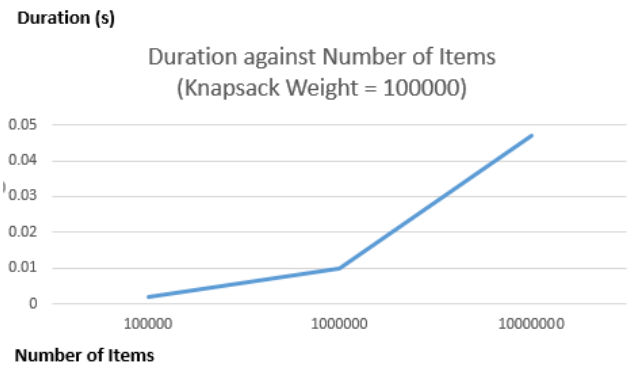
We can restructure the table as below:

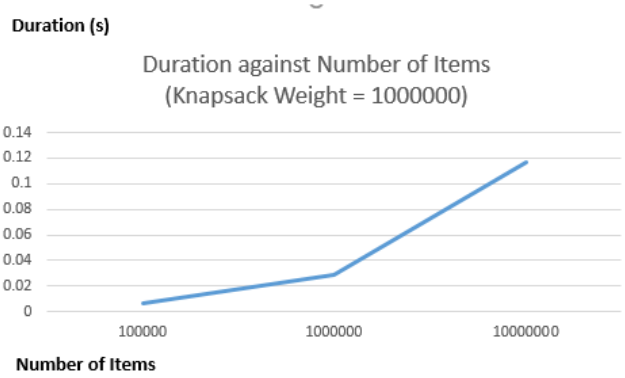
Number of Items \ Knapsack Weight	100000	1000000	10000000
100000	0.002031	0.007005	0.027014
1000000	0.010036	0.029022	0.105070
10000000	0.047032	0.117083	0.400309

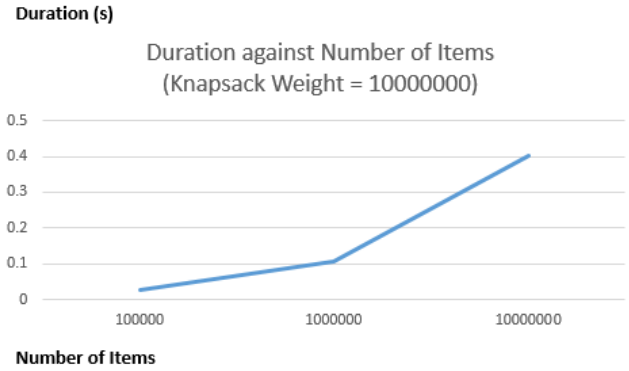
Best Case – Least number of items and lightest knapsack weight.
Average Case – Moderate number of items and moderate knapsack weight.
Worst Case – Most number of items and heaviest knapsack weight.

Let's plot duration against number of items and duration against knapsack weight to see the correlations.

Duration against Number of Items



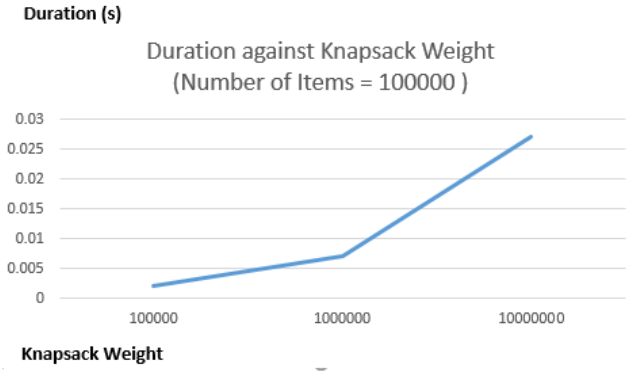


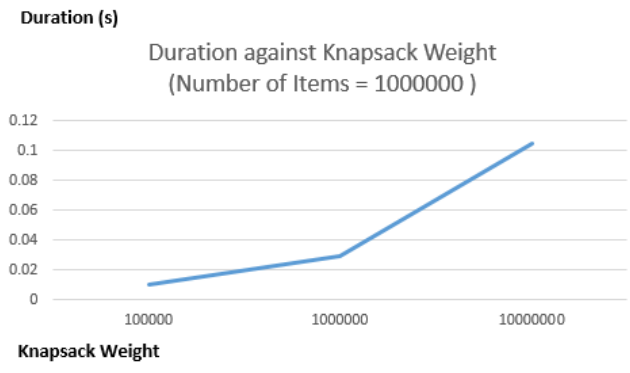


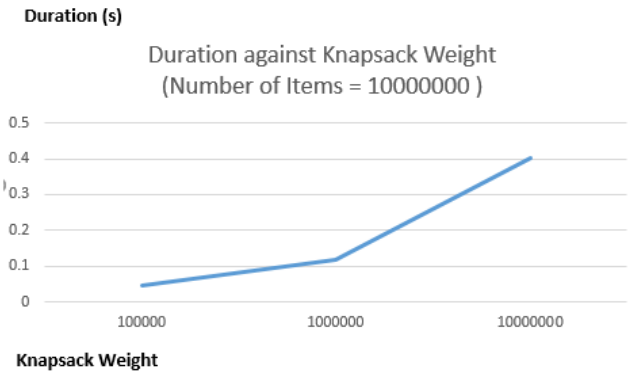
We can see that when the number of items increases, the duration (in seconds) increases as well.

This is because there are **more items to put** inside the knapsack.

Duration against Knapsack Weight







We can see that when the knapsack weight increases, the duration (in seconds) increases as well.

This is because you **can fit more items** in the knapsack.

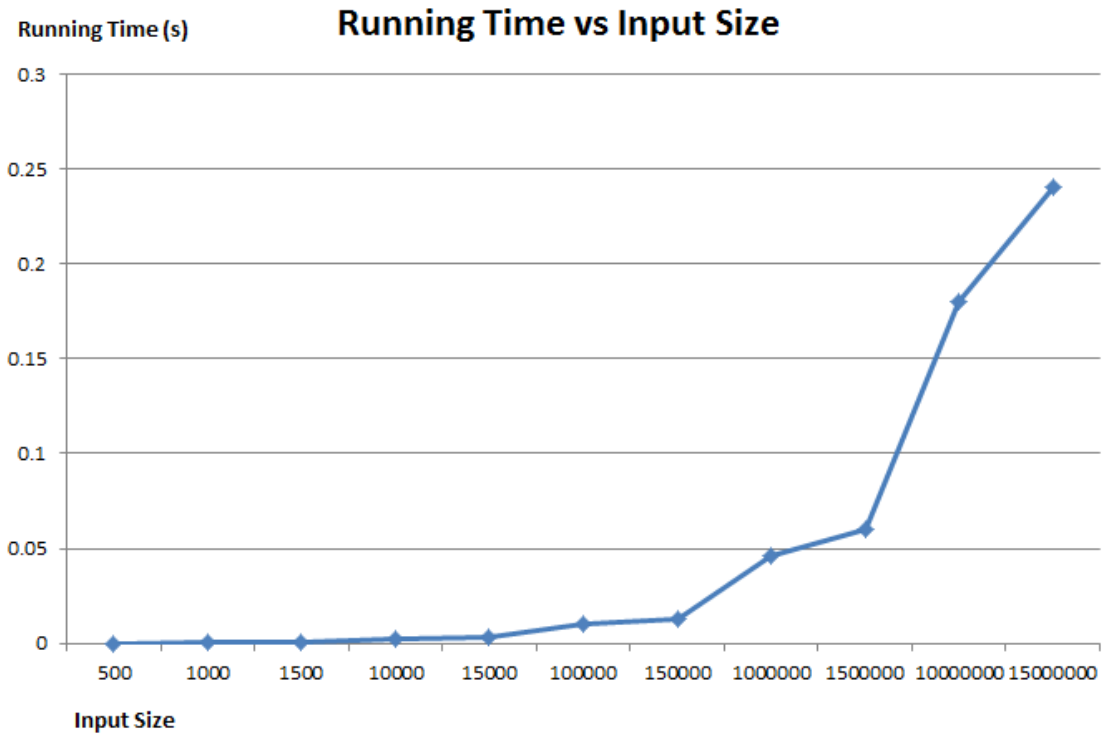
Running Time against Input Size

Knapsack Weight = 2,000,0000

Max Item Weight = 50,000,000

Max Item Benefit = 10,000,000

Number of Items	Duration (s)
500	0.000000
1000	0.001000
1500	0.001001
10000	0.002001
15000	0.003008
100000	0.010007
150000	0.013010
1000000	0.046034
1500000	0.060043
10000000	0.180127
15000000	0.240172



We can conclude that the fractional knapsack algorithm run in **O(n log n)** time.