MSHIMA001
Assignment 3:
This document is divided into 2 parts , answers to theoretical questions and traces and transcripts of unit tests and tail recursion.

1. **Answers to theoretical questions.**

2.2) f(n) = 6^n.
Reason: At each depth, the number of successive states increases by a factor of 6 hence after a depth of n , 6^n states will be generated.

3.2) To solve this question, the solveCube was called with a initial state of ((2 5) (1 1) (5 2) (4 1) (8 3) (3 4) (6 2) (7 3)) which will require 7 moves to solve ie "yZXzYYx"'

The following CPU and memory usage characteristics were observed.

- It took over 30 minutes to solve.
  The reason being that (6^7 + 6^6 + … 6^2 + 6) different states were generated and each of these had to compared with solvedStates. The complexity is of exponential order, hence explains the long duration taken.
- CPU usage increased by about 30%.
  This computation required a lot of CPU time hence explains the percentage increase in CPU time.
- Memory usage increased by about 2%.
  Memory usage didn't significantly increase because tail recursion was implemented for question 2.1. The reason being heap usage for tail-recursive functions is bounded by a constant (i.e., is O(1)).

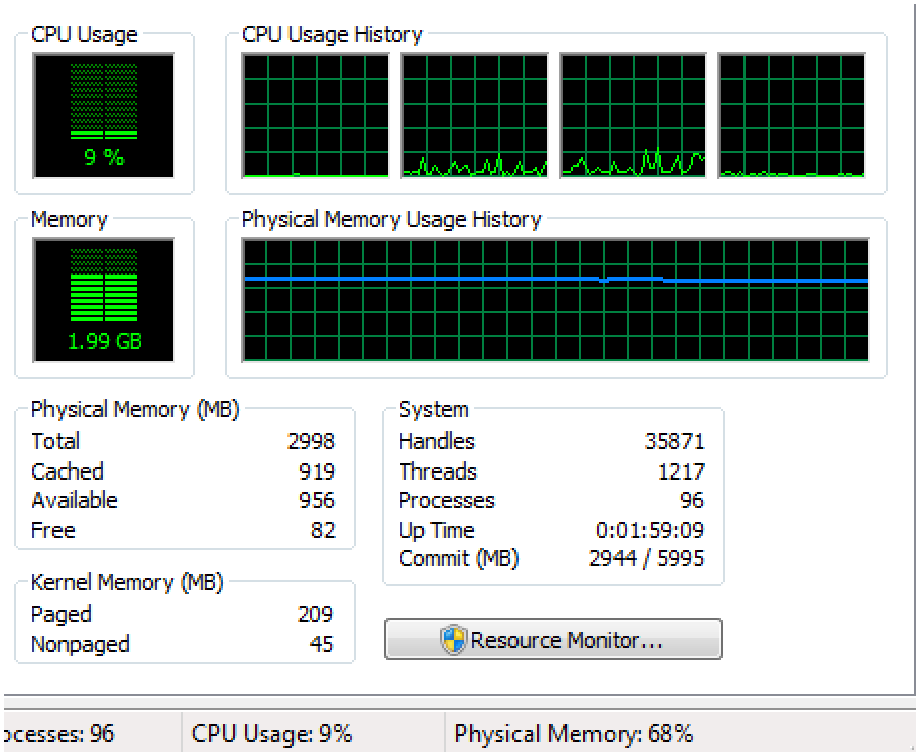

Fig 1: CPU and memory usage before running long function.

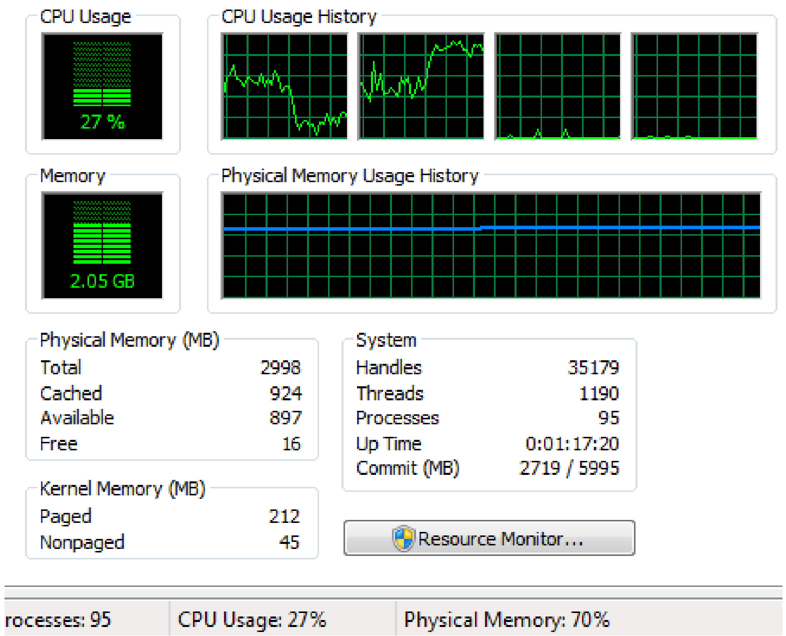

Fig 2:CPU and memory usage while running solveCube.

4) Since the number of successor states is reduced to 5. f(n) = 5^n. In a shuffle of 10 moves the total number of states generated is 5^1 + 5^2 + 5^3 + 5^4 + … + 5^10 in comparison, with 6 states this number is equal to 6^1 + 6^2 + 6^3 + ... + 6^10. The total percent in reduction is the ratio equal to difference between them and the original number.

%Decrease = 1-((5^1 + 5^2 + 5^3 +.... +5^10)/ (6^1 + 6^2 + 6^3 + ... + 6^10))
= 1 -(5* (5^10 -1)/4)/(6* (6^10-1)/5)
= 83.177%

This is equivalent to stating the computational cost decreased by a factor of 0.16823. Further optimisation can be done with a more efficient algorithm.

2. TRACES AND TRANSCRIPTS OF UNIT TESTS AND TAIL RECURSION.

2.1 UNIT TESTS

Similar to the unit tests provided in the skeleton code, I have provided more unit tests for the functions I defined. The following output was obtained when the program was loaded indicating that all tests passed. A similar output can be obtained by uncommenting the tests in the program.

```
;; TEST

(print (equal? (loopsolved 0 (genStates 0 original '()))  0) "\n")
(print (equal? (loopsolved 0 (genStates 1 original '()))  6) "\n")
```

Fig 3: sections of code with unit tests , uncomment them for proof.

```
7> (load "assignment3.scm")
#t
#t
#t
#t
#t
#t
#t
#t
#t
#t
#t
"C:\\Users\\DELL\\Desktop\\Scheme\\ass3-automarker-master\\assignment3.scm"
```

Fig 4: Evidence of unit testing.

<u>2.2 TAIL RECURSION</u>

Where needed , tail recursion for looping , this came as a result that head recursion results into heap overflow. An accumulator was used in the arguments of the functions. Functions that used tail recursion include:

- genHelper.
- loopresult
- findx
- loopsolved
- solvecube

By tracing genStates and solvecube the following transcripts were observed.

```
> (genStates 2 original '())
> (genHelper 2 '((((1 1) (2 1) (3 1) (4 1) (5 3) (6 3) (7 3) (8 3))) (())))
| > (loopresult 0 1 '(() ()) '((((1 1) (2 1) (3 1) (4 1) (5 3) (6 3) (7 3) ...

| > (loopresult 1 1 '((((5 4) (2 1) (1 2) (4 1) (7 4) (6 3) (3 2) (8 3)) ((...

| ((((5 4) (2 1) (1 2) (4 1) (7 4) (6 3) (3 2) (8 3)) ((3 4) (2 1) (7 2) (4...

> (genHelper 1 '((((5 4) (2 1) (1 2) (4 1) (7 4) (6 3) (3 2) (8 3)) ((3 4) ...

| > (loopresult 0 6 '(() ()) '((((5 4) (2 1) (1 2) (4 1) (7 4) (6 3) (3 2) ...

| > (loopresult 1 6 '((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((...

| > (loopresult 2 6 '((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((...

| > (loopresult 3 6 '((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((...

| > (loopresult 4 6 '((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((...

| > (loopresult 5 6 '((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((...

| > (loopresult 6 6 '((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((...

| ((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((1 1) (2 1) (3 1) (4...

> (genHelper 0 '((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((1 1) ...

((((7 1) (2 1) (5 1) (4 1) (3 3) (6 3) (1 3) (8 3)) ((1 1) (2 1) (3 1) (4 1...
```

Fig 5: Evidence of tail recursion with genStates.

```
> (solveCube solvedStates (rotate "xy" original) 0)
¦ > (solveCube '(((1 1) (2 1) (3 1) (4 1) (5 3) (6 3) (7 3) (8 3)) ((3 1) (1 ...

¦ > (solveCube '(((1 1) (2 1) (3 1) (4 1) (5 3) (6 3) (7 3) (8 3)) ((3 1) (1 ...

¦ ¦ > (loopresult 0 1 '(() ()) '((((5 4) (2 1) (1 2) (4 1) (6 3) (8 3) (7 5) ...

¦ ¦ > (loopresult 1 1 '((((6 4) (2 1) (5 1) (4 1) (7 5) (8 3) (1 3) (3 6)) ((...

¦ ¦ ((((6 4) (2 1) (5 1) (4 1) (7 5) (8 3) (1 3) (3 6)) ((1 1) (2 1) (7 5) (4...

¦ > (solveCube '(((1 1) (2 1) (3 1) (4 1) (5 3) (6 3) (7 3) (8 3)) ((3 1) (1 ...

¦ ¦ > (loopresult 0 1 '(() ()) '((((5 4) (2 1) (1 2) (4 1) (6 3) (8 3) (7 5) ...

¦ ¦ > (loopresult 1 1 '((((6 4) (2 1) (5 1) (4 1) (7 5) (8 3) (1 3) (3 6)) ((...

¦ ¦ ((((6 4) (2 1) (5 1) (4 1) (7 5) (8 3) (1 3) (3 6)) ((1 1) (2 1) (7 5) (4...

¦ ¦ > (loopresult 0 6 '(() ()) '((((6 4) (2 1) (5 1) (4 1) (7 5) (8 3) (1 3) ...

¦ ¦ > (loopresult 1 6 '((((7 5) (2 1) (6 1) (4 1) (1 4) (8 3) (5 2) (3 6)) ((...

¦ ¦ > (loopresult 2 6 '((((7 5) (2 1) (6 1) (4 1) (1 4) (8 3) (5 2) (3 6)) ((...

¦ ¦ > (loopresult 3 6 '((((7 5) (2 1) (6 1) (4 1) (1 4) (8 3) (5 2) (3 6)) ((...

¦ ¦ > (loopresult 4 6 '((((7 5) (2 1) (6 1) (4 1) (1 4) (8 3) (5 2) (3 6)) ((...

¦ ¦ > (loopresult 5 6 '((((7 5) (2 1) (6 1) (4 1) (1 4) (8 3) (5 2) (3 6)) ((...

¦ ¦ > (loopresult 6 6 '((((7 5) (2 1) (6 1) (4 1) (1 4) (8 3) (5 2) (3 6)) ((...

¦ ¦ ((((7 5) (2 1) (6 1) (4 1) (1 4) (8 3) (5 2) (3 6)) ((5 4) (2 1) (1 2) (4...

¦ ("Y" "X")
```

Fig 6: Evidence of tail recursion with solveCube.