**Assignment #1**

**COSC 3320 Mon-Fri 14:00-16:00 PM**

**Student: Xin Zhou**

**PSID: 1367925**

**Prof: Dr. Leiss**

**June 15th, 2016**

//Program Code in C++

#include "tower.h"

tower::tower() {

current = 1;

n = 6;

for (int j = 0; j < 12; j++) {

toDest[j] = 1;

}

for (int i = 1; i <= n; i++) {

toDest[i] = 0;

}

r = n;

start(n, "Start", "Aux1", "Aux3", "Aux2", "Aux4", "Dest", current);

}

void tower::start(int numberOfDisks, string start, string source, string dest, string aux, string last, string last1, int current)

{

move(1, start, source, current);

current++;

algorithm(n, "Start", "Aux1", "Aux3", "Aux2", "Aux4","Dest", current);

move(1, dest, last, current);

current++;

move(1, last, last1, current);

current++;

}

int tower::algorithm(int numberOfDisks, string start, string source, string dest, string aux, string last, string last1, int& current)

{

if (numberOfDisks == 1) {

move(numberOfDisks, source, aux, current);

current++;

move(numberOfDisks, aux, dest, current);

current++;

}

else if (numberOfDisks >= 2) {

current = algorithm(numberOfDisks - 1, start, source, dest, aux, last, last1, current);

if (hasMoved[numberOfDisks] != 1) {

move(numberOfDisks, start, source, current);

current++;

hasMoved[numberOfDisks] = 1;

}

move(numberOfDisks, source, aux, current);

current++;

current = algorithm(numberOfDisks - 1, start, dest, source, aux, last, last1, current);

move(numberOfDisks, aux, dest, current);

current++;

if (toDest[numberOfDisks + 1] != 0) {

move(numberOfDisks, dest, last, current);

current++;

move(numberOfDisks, last, last1, current);

current++;

toDest[numberOfDisks] = 1;

}

if (numberOfDisks == r)

r--;

current = algorithm(numberOfDisks - 1, start, source, dest, aux, last, last1, current);

};

return current;

}

void tower::move(int adisk, string source, string dest, int current)

{

cout << "Move " << current << ": Move disk# " << adisk << " from " << source << " to " << dest << endl;

}

tower::~tower()

{

}

#pragma once

#include<iostream>

#include<string>

using namespace std;

class tower {

public:

tower();

~tower();

void start(int numofdisk, string start, string source, string dest, string aux, string last, string last1, int current);

int algorithm(int numofdisk, string start, string source, string dest, string aux, string last, string last1, int& current);

void move(int adisk, string source, string dest, int current);

private:

int n, current, r;

int hasMoved[12], toDest[12];

};

#include<iostream>

#include<time.h>

#include "tower.h"

using namespace std;

int main() {

clock\_t begin = clock();

tower to;

clock\_t end = clock();

cout << "Time is: " << (end - begin) << " ms" << endl;

system("pause");

return 0;

}

**Output:**

N = 1:

Move 1: Move disk# 1 from Start to Aux1

Move 2: Move disk# 1 from Aux1 to Aux2

Move 3: Move disk# 1 from Aux2 to Aux3

Move 4: Move disk# 1 from Aux3 to Aux4

Move 5: Move disk# 1 from Aux4 to Dest

N = 2:

Move 1: Move disk# 1 from Start to Aux1

Move 2: Move disk# 1 from Aux1 to Aux2

Move 3: Move disk# 1 from Aux2 to Aux3

Move 4: Move disk# 2 from Start to Aux1

Move 5: Move disk# 2 from Aux1 to Aux2

Move 6: Move disk# 1 from Aux3 to Aux2

Move 7: Move disk# 1 from Aux2 to Aux1

Move 8: Move disk# 2 from Aux2 to Aux3

Move 9: Move disk# 2 from Aux3 to Aux4

Move 10: Move disk# 2 from Aux4 to Dest

Move 11: Move disk# 1 from Aux1 to Aux2

Move 12: Move disk# 1 from Aux2 to Aux3

Move 13: Move disk# 1 from Aux3 to Aux4

Move 14: Move disk# 1 from Aux4 to Dest

N = 3:

Move 1: Move disk# 1 from Start to Aux1

Move 2: Move disk# 1 from Aux1 to Aux2

Move 3: Move disk# 1 from Aux2 to Aux3

Move 4: Move disk# 2 from Start to Aux1

Move 5: Move disk# 2 from Aux1 to Aux2

Move 6: Move disk# 1 from Aux3 to Aux2

Move 7: Move disk# 1 from Aux2 to Aux1

Move 8: Move disk# 2 from Aux2 to Aux3

Move 9: Move disk# 1 from Aux1 to Aux2

Move 10: Move disk# 1 from Aux2 to Aux3

Move 11: Move disk# 3 from Start to Aux1

Move 12: Move disk# 3 from Aux1 to Aux2

Move 13: Move disk# 1 from Aux3 to Aux2

Move 14: Move disk# 1 from Aux2 to Aux1

Move 15: Move disk# 2 from Aux3 to Aux2

Move 16: Move disk# 1 from Aux1 to Aux2

Move 17: Move disk# 1 from Aux2 to Aux3

Move 18: Move disk# 2 from Aux2 to Aux1

Move 19: Move disk# 1 from Aux3 to Aux2

Move 20: Move disk# 1 from Aux2 to Aux1

Move 21: Move disk# 3 from Aux2 to Aux3

Move 22: Move disk# 3 from Aux3 to Aux4

Move 23: Move disk# 3 from Aux4 to Dest

Move 24: Move disk# 1 from Aux1 to Aux2

Move 25: Move disk# 1 from Aux2 to Aux3

Move 26: Move disk# 2 from Aux1 to Aux2

Move 27: Move disk# 1 from Aux3 to Aux2

Move 28: Move disk# 1 from Aux2 to Aux1

Move 29: Move disk# 2 from Aux2 to Aux3

Move 30: Move disk# 2 from Aux3 to Aux4

Move 31: Move disk# 2 from Aux4 to Dest

Move 32: Move disk# 1 from Aux1 to Aux2

Move 33: Move disk# 1 from Aux2 to Aux3

Move 34: Move disk# 1 from Aux3 to Aux4

Move 35: Move disk# 1 from Aux4 to Dest

N = 4 :

Move 1: Move disk# 1 from Start to Aux1

Move 2: Move disk# 1 from Aux1 to Aux2

Move 3: Move disk# 1 from Aux2 to Aux3

Move 4: Move disk# 2 from Start to Aux1

Move 5: Move disk# 2 from Aux1 to Aux2

Move 6: Move disk# 1 from Aux3 to Aux2

Move 7: Move disk# 1 from Aux2 to Aux1

Move 8: Move disk# 2 from Aux2 to Aux3

Move 9: Move disk# 1 from Aux1 to Aux2

Move 10: Move disk# 1 from Aux2 to Aux3

Move 11: Move disk# 3 from Start to Aux1

Move 12: Move disk# 3 from Aux1 to Aux2

Move 13: Move disk# 1 from Aux3 to Aux2

Move 14: Move disk# 1 from Aux2 to Aux1

Move 15: Move disk# 2 from Aux3 to Aux2

Move 16: Move disk# 1 from Aux1 to Aux2

Move 17: Move disk# 1 from Aux2 to Aux3

Move 18: Move disk# 2 from Aux2 to Aux1

Move 19: Move disk# 1 from Aux3 to Aux2

Move 20: Move disk# 1 from Aux2 to Aux1

Move 21: Move disk# 3 from Aux2 to Aux3

Move 22: Move disk# 1 from Aux1 to Aux2

Move 23: Move disk# 1 from Aux2 to Aux3

Move 24: Move disk# 2 from Aux1 to Aux2

Move 25: Move disk# 1 from Aux3 to Aux2

Move 26: Move disk# 1 from Aux2 to Aux1

Move 27: Move disk# 2 from Aux2 to Aux3

Move 28: Move disk# 1 from Aux1 to Aux2

Move 29: Move disk# 1 from Aux2 to Aux3

Move 30: Move disk# 4 from Start to Aux1

Move 31: Move disk# 4 from Aux1 to Aux2

Move 32: Move disk# 1 from Aux3 to Aux2

Move 33: Move disk# 1 from Aux2 to Aux1

Move 34: Move disk# 2 from Aux3 to Aux2

Move 35: Move disk# 1 from Aux1 to Aux2

Move 36: Move disk# 1 from Aux2 to Aux3

Move 37: Move disk# 2 from Aux2 to Aux1

Move 38: Move disk# 1 from Aux3 to Aux2

Move 39: Move disk# 1 from Aux2 to Aux1

Move 40: Move disk# 3 from Aux3 to Aux2

Move 41: Move disk# 1 from Aux1 to Aux2

Move 42: Move disk# 1 from Aux2 to Aux3

Move 43: Move disk# 2 from Aux1 to Aux2

Move 44: Move disk# 1 from Aux3 to Aux2

Move 45: Move disk# 1 from Aux2 to Aux1

Move 46: Move disk# 2 from Aux2 to Aux3

Move 47: Move disk# 1 from Aux1 to Aux2

Move 48: Move disk# 1 from Aux2 to Aux3

Move 49: Move disk# 3 from Aux2 to Aux1

Move 50: Move disk# 1 from Aux3 to Aux2

Move 51: Move disk# 1 from Aux2 to Aux1

Move 52: Move disk# 2 from Aux3 to Aux2

Move 53: Move disk# 1 from Aux1 to Aux2

Move 54: Move disk# 1 from Aux2 to Aux3

Move 55: Move disk# 2 from Aux2 to Aux1

Move 56: Move disk# 1 from Aux3 to Aux2

Move 57: Move disk# 1 from Aux2 to Aux1

Move 58: Move disk# 4 from Aux2 to Aux3

Move 59: Move disk# 4 from Aux3 to Aux4

Move 60: Move disk# 4 from Aux4 to Dest

Move 61: Move disk# 1 from Aux1 to Aux2

Move 62: Move disk# 1 from Aux2 to Aux3

Move 63: Move disk# 2 from Aux1 to Aux2

Move 64: Move disk# 1 from Aux3 to Aux2

Move 65: Move disk# 1 from Aux2 to Aux1

Move 66: Move disk# 2 from Aux2 to Aux3

Move 67: Move disk# 1 from Aux1 to Aux2

Move 68: Move disk# 1 from Aux2 to Aux3

Move 69: Move disk# 3 from Aux1 to Aux2

Move 70: Move disk# 1 from Aux3 to Aux2

Move 71: Move disk# 1 from Aux2 to Aux1

Move 72: Move disk# 2 from Aux3 to Aux2

Move 73: Move disk# 1 from Aux1 to Aux2

Move 74: Move disk# 1 from Aux2 to Aux3

Move 75: Move disk# 2 from Aux2 to Aux1

Move 76: Move disk# 1 from Aux3 to Aux2

Move 77: Move disk# 1 from Aux2 to Aux1

Move 78: Move disk# 3 from Aux2 to Aux3

Move 79: Move disk# 3 from Aux3 to Aux4

Move 80: Move disk# 3 from Aux4 to Dest

Move 81: Move disk# 1 from Aux1 to Aux2

Move 82: Move disk# 1 from Aux2 to Aux3

Move 83: Move disk# 2 from Aux1 to Aux2

Move 84: Move disk# 1 from Aux3 to Aux2

Move 85: Move disk# 1 from Aux2 to Aux1

Move 86: Move disk# 2 from Aux2 to Aux3

Move 87: Move disk# 2 from Aux3 to Aux4

Move 88: Move disk# 2 from Aux4 to Dest

Move 89: Move disk# 1 from Aux1 to Aux2

Move 90: Move disk# 1 from Aux2 to Aux3

Move 91: Move disk# 1 from Aux3 to Aux4

Move 92: Move disk# 1 from Aux4 to Dest

N = 5, 6, 7, 8, 9, 10; when N >= 5 ,

**They have the same first 100 moves and last 100 moves**

Move 1: Move disk# 1 from Start to Aux1

Move 2: Move disk# 1 from Aux1 to Aux2

Move 3: Move disk# 1 from Aux2 to Aux3

Move 4: Move disk# 2 from Start to Aux1

Move 5: Move disk# 2 from Aux1 to Aux2

Move 6: Move disk# 1 from Aux3 to Aux2

Move 7: Move disk# 1 from Aux2 to Aux1

Move 8: Move disk# 2 from Aux2 to Aux3

Move 9: Move disk# 1 from Aux1 to Aux2

Move 10: Move disk# 1 from Aux2 to Aux3

Move 11: Move disk# 3 from Start to Aux1

Move 12: Move disk# 3 from Aux1 to Aux2

Move 13: Move disk# 1 from Aux3 to Aux2

Move 14: Move disk# 1 from Aux2 to Aux1

Move 15: Move disk# 2 from Aux3 to Aux2

Move 16: Move disk# 1 from Aux1 to Aux2

Move 17: Move disk# 1 from Aux2 to Aux3

Move 18: Move disk# 2 from Aux2 to Aux1

Move 19: Move disk# 1 from Aux3 to Aux2

Move 20: Move disk# 1 from Aux2 to Aux1

Move 21: Move disk# 3 from Aux2 to Aux3

Move 22: Move disk# 1 from Aux1 to Aux2

Move 23: Move disk# 1 from Aux2 to Aux3

Move 24: Move disk# 2 from Aux1 to Aux2

Move 25: Move disk# 1 from Aux3 to Aux2

Move 26: Move disk# 1 from Aux2 to Aux1

Move 27: Move disk# 2 from Aux2 to Aux3

Move 28: Move disk# 1 from Aux1 to Aux2

Move 29: Move disk# 1 from Aux2 to Aux3

Move 30: Move disk# 4 from Start to Aux1

Move 31: Move disk# 4 from Aux1 to Aux2

Move 32: Move disk# 1 from Aux3 to Aux2

Move 33: Move disk# 1 from Aux2 to Aux1

Move 34: Move disk# 2 from Aux3 to Aux2

Move 35: Move disk# 1 from Aux1 to Aux2

Move 36: Move disk# 1 from Aux2 to Aux3

Move 37: Move disk# 2 from Aux2 to Aux1

Move 38: Move disk# 1 from Aux3 to Aux2

Move 39: Move disk# 1 from Aux2 to Aux1

Move 40: Move disk# 3 from Aux3 to Aux2

Move 41: Move disk# 1 from Aux1 to Aux2

Move 42: Move disk# 1 from Aux2 to Aux3

Move 43: Move disk# 2 from Aux1 to Aux2

Move 44: Move disk# 1 from Aux3 to Aux2

Move 45: Move disk# 1 from Aux2 to Aux1

Move 46: Move disk# 2 from Aux2 to Aux3

Move 47: Move disk# 1 from Aux1 to Aux2

Move 48: Move disk# 1 from Aux2 to Aux3

Move 49: Move disk# 3 from Aux2 to Aux1

Move 50: Move disk# 1 from Aux3 to Aux2

Move 51: Move disk# 1 from Aux2 to Aux1

Move 52: Move disk# 2 from Aux3 to Aux2

Move 53: Move disk# 1 from Aux1 to Aux2

Move 54: Move disk# 1 from Aux2 to Aux3

Move 55: Move disk# 2 from Aux2 to Aux1

Move 56: Move disk# 1 from Aux3 to Aux2

Move 57: Move disk# 1 from Aux2 to Aux1

Move 58: Move disk# 4 from Aux2 to Aux3

Move 59: Move disk# 1 from Aux1 to Aux2

Move 60: Move disk# 1 from Aux2 to Aux3

Move 61: Move disk# 2 from Aux1 to Aux2

Move 62: Move disk# 1 from Aux3 to Aux2

Move 63: Move disk# 1 from Aux2 to Aux1

Move 64: Move disk# 2 from Aux2 to Aux3

Move 65: Move disk# 1 from Aux1 to Aux2

Move 66: Move disk# 1 from Aux2 to Aux3

Move 67: Move disk# 3 from Aux1 to Aux2

Move 68: Move disk# 1 from Aux3 to Aux2

Move 69: Move disk# 1 from Aux2 to Aux1

Move 70: Move disk# 2 from Aux3 to Aux2

Move 71: Move disk# 1 from Aux1 to Aux2

Move 72: Move disk# 1 from Aux2 to Aux3

Move 73: Move disk# 2 from Aux2 to Aux1

Move 74: Move disk# 1 from Aux3 to Aux2

Move 75: Move disk# 1 from Aux2 to Aux1

Move 76: Move disk# 3 from Aux2 to Aux3

Move 77: Move disk# 1 from Aux1 to Aux2

Move 78: Move disk# 1 from Aux2 to Aux3

Move 79: Move disk# 2 from Aux1 to Aux2

Move 80: Move disk# 1 from Aux3 to Aux2

Move 81: Move disk# 1 from Aux2 to Aux1

Move 82: Move disk# 2 from Aux2 to Aux3

Move 83: Move disk# 1 from Aux1 to Aux2

Move 84: Move disk# 1 from Aux2 to Aux3

Move 85: Move disk# 5 from Start to Aux1

Move 86: Move disk# 5 from Aux1 to Aux2

Move 87: Move disk# 1 from Aux3 to Aux2

Move 88: Move disk# 1 from Aux2 to Aux1

Move 89: Move disk# 2 from Aux3 to Aux2

Move 90: Move disk# 1 from Aux1 to Aux2

Move 91: Move disk# 1 from Aux2 to Aux3

Move 92: Move disk# 2 from Aux2 to Aux1

Move 93: Move disk# 1 from Aux3 to Aux2

Move 94: Move disk# 1 from Aux2 to Aux1

Move 95: Move disk# 3 from Aux3 to Aux2

Move 96: Move disk# 1 from Aux1 to Aux2

Move 97: Move disk# 1 from Aux2 to Aux3

Move 98: Move disk# 2 from Aux1 to Aux2

Move 99: Move disk# 1 from Aux3 to Aux2

Move 100: Move disk# 1 from Aux2 to Aux1

Last 100 moves:

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 3 from Aux2 to Aux1

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux2 to Aux1

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 5 from Aux2 to Aux3

Last 100 Moves : Move disk# 5 from Aux3 to Aux4

Last 100 Moves : Move disk# 5 from Aux4 to Dest

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux2 to Aux3

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 3 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux2 to Aux1

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 3 from Aux2 to Aux3

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux2 to Aux3

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 4 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux2 to Aux1

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 3 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux2 to Aux3

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 3 from Aux2 to Aux1

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux2 to Aux1

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 4 from Aux2 to Aux3

Last 100 Moves : Move disk# 4 from Aux3 to Aux4

Last 100 Moves : Move disk# 4 from Aux4 to Dest

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux2 to Aux3

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 3 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux2 to Aux1

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 3 from Aux2 to Aux3

Last 100 Moves : Move disk# 3 from Aux3 to Aux4

Last 100 Moves : Move disk# 3 from Aux4 to Dest

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux3 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux1

Last 100 Moves : Move disk# 2 from Aux2 to Aux3

Last 100 Moves : Move disk# 2 from Aux3 to Aux4

Last 100 Moves : Move disk# 2 from Aux4 to Dest

Last 100 Moves : Move disk# 1 from Aux1 to Aux2

Last 100 Moves : Move disk# 1 from Aux2 to Aux3

Last 100 Moves : Move disk# 1 from Aux3 to Aux4

Last 100 Moves : Move disk# 1 from Aux4 to Dest

**3.**

**a) Execute optimally (that is A[lo:m] and A[m:hi] are always of equal size)**

If the Quicksort algorithm is optimal, then the pivot needs to split data equally.

That means we can let pivot be the median number (choose element to the right of two middle numbers if n is even) of the given sequence recursively. Therefore, we need to split log2(n) times totally and compare n-1 times for every split.

--- When n is infinite, the time complexity is O(n\*log2(n)).

For example n = 5:

N (5) = {5, 1, 3, 4, 2}, we choose the median number 3 as the pivot.

So, it needs to split log2(n) = 3 times and compare n-1 times each loop.

(5 – 1) \* 3 = 12 comparisons

**b) Execute in the slowest possible way.**

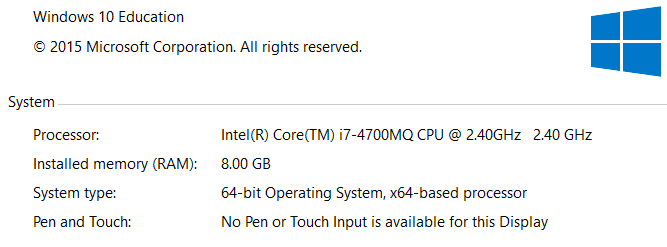
The slowest possible way is that we choose pivot from 2nd element or penultimate element of the array. That means the Quicksort needs to split n-1 times and compare n-1 times each loop. (n-1)(n-1)= n^2- 2n +1. Therefore we get big O notation is O(n^2) if n is infinite.

|  |  |  |  |
| --- | --- | --- | --- |
| Quicksort | The Best Case | Average Case | The Worst Case |
| Time | O(n\*log2(n)) | O(n\*log2(n)) | O(n^2) |

4)

Memory Fragmentation in C++ was performed in my Lenovo laptop.

The following screenshot has its specification:



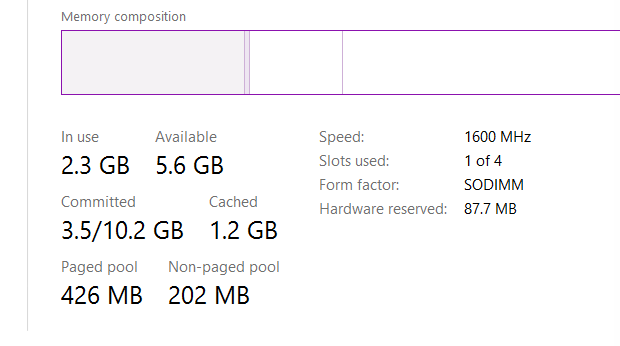
At first, I ran my program for many times, but it never exhausted all of the main memory of my laptop. So, I was thinking 64bit-Window 10 Education cannot take full advantage of 8GB of RAM. Then, I found the information about memory limit in the different systems. Unfortunately, the program should be able to exhaust all of the main memory available to my program. After I checked the values of all variables and debugged the code error, I still had the error which was “stack overflow” even the value of “M” is pretty small. Therefore, I found that I had to change the stack reserve size and stack commit size to be large enough to exhaust all of the main memory. After I changed that, my program rans well.

When I put M = 500, my program rans as normal.

When I put M = 5000, the main memory was going up to 100% in use after few seconds and my laptop crashed.

After I restarted my laptop, I calculated the value of “M” theoretically.

Available Main Memory was about 5.6 GB when I didn’t run my program.



Then I should be able to find the value of “M” which could exhaust almost all of the main memory available to my program.

**5.6 GB = 5.6 \*1024 \*1024 \*1024 = 6012954214 bytes**

**Since every element in a 2D array occupying one word or 4 bytes.**

**Thus, M = 6012954214 / 4 bytes per element / 3 / 800,000 = 626.**

Therefore, I used M = 630 that is a little larger than 626 which exhausted all available main memory to my program for few seconds.

Output is:

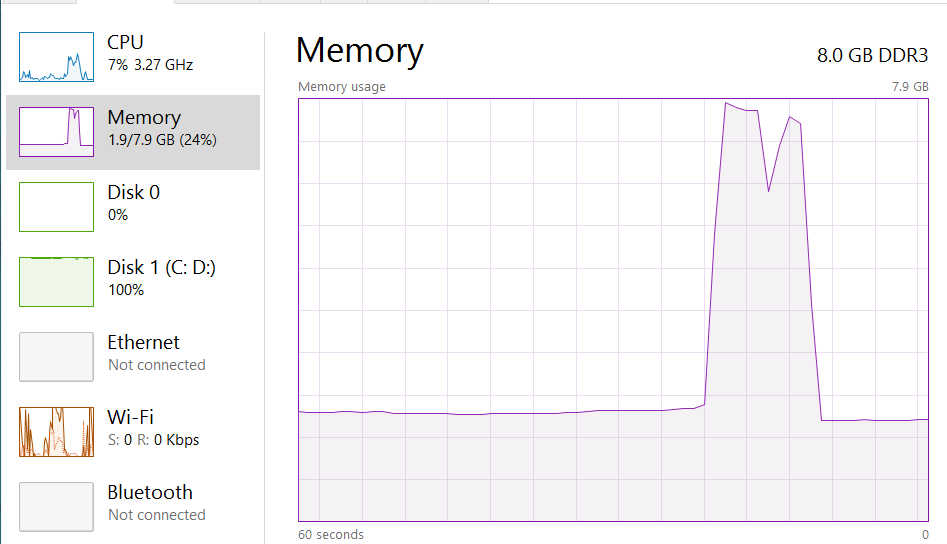
![C:\Users\XinZhou\AppData\Roaming\Tencent\Users\461548602\QQ\WinTemp\RichOle\5ZBIX8EP@CTWKIEMGBL](8S.png](data:image/png;base64,)

.

Allocating memory for first sequence of 3M arrays needs: 5651 ms

Deallocating all even-numbered arrays needs: 716 ms (Garbage Collection)

Allocating a sequence of M arrays needs: 2607 ms



#include <iostream>

#include <time.h>

#include <algorithm>

#define M 630

using namespace std;

int main() {

clock\_t begin1, stop1, begin2, stop2, begin3, stop3;

double tdif = 0, tdif2 = 0, tdif3 = 0;

int \*array1[3 \* M], \*array2[M], i, j;

double dif, dif2, dif3;

begin1 = clock();

cout << " Step 1: Allocating memory for first sequence of 3M arrays of size 800,000 elements each..." << endl;

for (i = 0; i < 3 \* M; i++)

array1[i] = (int\*)malloc(800000 \* sizeof(int));

for (i = 0; i < 3 \* M; i++)

{

for (j = 0; j < 800000; j++)

array1[i][j] = j;//allocating 1st arrays

}

stop1 = clock();

begin2 = clock();

cout << "Step 2: Deallocating all even-numbered arrays of first squences of 3M arrays..." << endl;

// delete even number array in array1

for (i = 0; i < 3 \* M; i += 2) {

delete(array1[i]);

}

stop2 = clock();

begin3 = clock();

cout << " Step 3: Allocaintg a squence of M arrays of size 1,200,000 elements each..." << endl;

for (i = 0; i < M; i++)

array2[i] = (int\*)malloc(1200000 \* sizeof(int));

for (i = 0; i < M; i++)

{

for (j = 0; j < 1200000; j++)

array2[i][j] = j;

}

stop3 = clock();

dif = (stop1 - begin1);

dif2 = (stop2 - begin2);

dif3 = (stop3 - begin3);

cout << "Step 4: Deallocating all arrays ..." << endl;

for (int i = 1; i < 3 \* M; i += 2) {

delete(array1[i]);

}

for (int i = 1; i < M; i++) {

delete(array2[i]);

}

cout << "Allocating memory for first sequence of 3M arrays needs: " <<dif << " ms" << endl;

cout << "Deallocating all even-numbered arrays needs: " << dif2 << " ms" << endl;

cout << "Allocaintg a squence of M arrays needs: " << dif3 << " ms" << endl;

system("pause");

return 0;

}

**5) Carry out the same 10,000,000 unsuccessful searches for eight different-sized arrays, namely arrays of size 128, 512, 2048, 8192, 32768, 131072, 524288, and 2097152.**

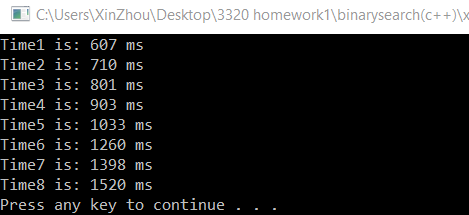
|  |  |  |
| --- | --- | --- |
| Best case | Average | Worst case |
| O(1) | O (log2 (n)). | O (log2 (n)). |

The time complexity of binary search algorithm is O (log2 (n)) since we carry out the same 10,000,000 unsuccessful searches. Therefore, the worst case space complexity is O (log2 (n))

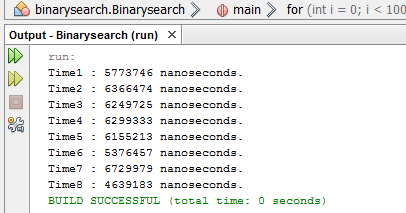
128 = 2^7; 512 = 2^9; 2048 = 2^11; 8192 = 2^13;

32768 = 2^15; 131072 = 2^17; 524288 = 2^19; 2097152 = 2^21

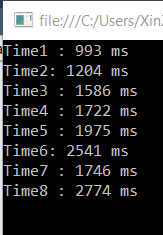
C++ program (Visual Studio):



Java (NetBeans):



C# (Visual Studio)



The performance of three languages and theoretical.

|  |  |  |  |
| --- | --- | --- | --- |
| Theoretically | C++ (Average) | Java (Average) | C# (Average) |
| 7 | 610 ms | 6.54 ms | 1003 ms |
| 9 | 705 ms | 6.37 ms | 1241 ms |
| 11 | 809 ms | 6.25 ms | 1591 ms |
| 13 | 915 ms | 5.31 ms | 1759 ms |
| 15 | 1045 ms | 5.30 ms | 1995 ms |
| 17 | 1250 ms | 5.24 ms | 2554 ms |
| 19 | 1392 ms | 5.08 ms | 1760 ms |
| 21 | 1526 ms | 4.79 ms | 2818 ms |

We can see their performances through the table and the chart above.

Java performed the best and C# performed the worst.

Before these tests, I thought C++ should be the fastest one because it is compiled high level language. In this experiment, JAVA is much faster than C++, C# and theoretical. We can see that the graph of C++ is close to the theoretical graph.

C++

#include <iostream>

#include<time.h>

using namespace std;

int binarySearch(int array[], int size, int item);

void fillArray(int array[], int n);

int main()

{

int a = 1;

int array1[128];

int array2[512];

int array3[2048];

int array4[8192];

int array5[32768];

int array6[131072];

int\* array7 = NULL;

int\* array8 = NULL;

array7 = new int[524288];

array8 = new int[2097152];

fillArray(array1, 128);

fillArray(array2, 512);

fillArray(array3, 2048);

fillArray(array4, 8192);

fillArray(array5, 32768);

fillArray(array6, 131072);

fillArray(array7, 524288);

fillArray(array8, 2097152);

clock\_t begin1 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array1, 128,-1);

}

clock\_t end1 = clock();

clock\_t begin2 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array2, 512, -1);

}

clock\_t end2 = clock();

clock\_t begin3 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array3, 2048, -1);

}

clock\_t end3 = clock();

clock\_t begin4 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array4, 8192, -1);

}

clock\_t end4 = clock();

clock\_t begin5 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array5, 32768, -1);

}

clock\_t end5 = clock();

clock\_t begin6 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array6, 131072, -1);

}

clock\_t end6 = clock();

clock\_t begin7 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array7, 524288, -1);

}

clock\_t end7 = clock();

clock\_t begin8 = clock();

for (int i = 0; i < 10000000; i++)

{

a = binarySearch(array8, 2097152, -1);

}

clock\_t end8 = clock();

cout << "Time1 is: " << (end1 - begin1) << " ms" << endl;

cout << "Time2 is: " << (end2 - begin2) << " ms" << endl;

cout << "Time3 is: " << (end3 - begin3) << " ms" << endl;

cout << "Time4 is: " << (end4 - begin4) << " ms" << endl;

cout << "Time5 is: " << (end5 - begin5) << " ms" << endl;

cout << "Time6 is: " << (end6 - begin6) << " ms" << endl;

cout << "Time7 is: " << (end7 - begin7) << " ms" << endl;

cout << "Time8 is: " << (end8 - begin8) << " ms" << endl;

system("pause");

return 0;

}

int binarySearch(int array[], int size, int item)

{

int first = 0;

int last = size - 1;

int mid;

bool found = false;

while (first <= last && !found)

{

mid = (first + last) / 2;

if (array[mid] == item)

found = true;

else if (array[mid] > item)

last = mid - 1;

else

first = mid + 1;

}

if (found)

return 0;

else

return -1;

}

void fillArray(int array[], int n) {

for (int i = 1; i <= n; i++) {

array[i] = i;

}

}

Java:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package binarysearch;

/\*\*

\*

\* @author Shichang

\*/

public class Binarysearch {

public static void main(String[] args) {

int[] array1 = new int[128];

int[] array2 = new int[512];

int[] array3 = new int[2048];

int[] array4 = new int[8192];

int[] array5 = new int[32768];

int[] array6 = new int[131072];

int[] array7 = new int[524288];

int[] array8 = new int[2097152];

int a;

for (int i = 0; i < 128; i++)

{

array1[i] = i;

}

for (int i = 0; i < 512; i++)

{

array2[i] = i;

}

for (int i = 0; i < 2048; i++)

{

array3[i] = i;

}

for (int i = 0; i < 8192; i++)

{

array4[i] = i;

}

for (int i = 0; i < 32768; i++)

{

array5[i] = i;

}

for (int i = 0; i < 131072; i++)

{

array6[i] = i;

}

for (int i = 0; i < 524288; i++)

{

array7[i] = i;

}

for (int i = 0; i < 2097152; i++)

{

array8[i] = i;

}

long startTime1 = System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array1,128,-1);

}

long stopTime1 = System.nanoTime();

System.out.println("Time1 : " + (stopTime1 - startTime1) + " nanoseconds.");

long startTime2 = System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array2,512,-1);

}

long stopTime2 = System.nanoTime();

System.out.println("Time2 : " + (stopTime2 - startTime2) + " nanoseconds.");

long startTime3 = System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array3,2048,-1);

}

long stopTime3 = System.nanoTime();

System.out.println("Time3 : " + (stopTime3 - startTime3) + " nanoseconds.");

long startTime4 = System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array4,8192,-1);

}

long stopTime4 = System.nanoTime();

System.out.println("Time4 : " + (stopTime4 - startTime4) + " nanoseconds.");

long startTime5 = System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array5,32768,-1);

}

long stopTime5 = System.nanoTime();

System.out.println("Time5 : " + (stopTime5 - startTime5) + " nanoseconds.");

long startTime6 = System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array6,131072,-1);

}

long stopTime6 = System.nanoTime();

System.out.println("Time6 : " + (stopTime6 - startTime6) + " nanoseconds.");

long startTime7 = System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array7,524588,-1);

}

long stopTime7 = System.nanoTime();

System.out.println("Time7 : " + (stopTime7 - startTime7) + " nanoseconds.");

long startTime8= System.nanoTime();

for(int i = 0; i < 10000000;i++){

a = binarySearch(array8,2097152,-1);

}

long stopTime8 = System.nanoTime();

System.out.println("Time8 : " + (stopTime8 - startTime8) + " nanoseconds.");

}

public static int binarySearch(int[] array, int size, int item)

{

int first = 0;

int last = size - 1;

int mid;

boolean found = false;

while (first <= last && !found){

mid = (first + last) / 2;

if (array[mid] == item)

if (array[mid] > item)

last = mid - 1;

else

first = mid + 1;

else found = true;

}

if (found)

return 0;

else

return -1;

}

}

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Timers;

using System.Diagnostics;

namespace ConsoleApplication1

{

class Program

{

private static int[] array = new int[128];

static void Main(string[] args)

{

Program prog = new Program();

int temp;

Stopwatch timer = new Stopwatch();

prog.FillArray(array, 128);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, -1, 128);

timer.Stop();

Console.WriteLine("Time1 : " + timer.ElapsedMilliseconds + " ms");

array = new int[512];

prog.FillArray(array, 512);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, -1, 512);

timer.Stop();

Console.WriteLine("Time2: " + timer.ElapsedMilliseconds + " ms");

array = new int[2048];

prog.FillArray(array, 2048);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, -1, 2048);

timer.Stop();

Console.WriteLine("Time3 : " + timer.ElapsedMilliseconds + " ms");

array = new int[8192];

prog.FillArray(array, 8192);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, -1, 8192);

timer.Stop();

Console.WriteLine("Time4 : " + timer.ElapsedMilliseconds + " ms");

array = new int[32768];

prog.FillArray(array, 32768);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, -1, 32768);

timer.Stop();

Console.WriteLine("Time5 : " + timer.ElapsedMilliseconds + " ms");

array = new int[131072];

prog.FillArray(array, 131072);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, i, 131072);

timer.Stop();

Console.WriteLine("Time6: " + timer.ElapsedMilliseconds + " ms");

array = new int[524288];

prog.FillArray(array, 524288);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, -1, 8192);

timer.Stop();

Console.WriteLine("Time7 : " + timer.ElapsedMilliseconds + " ms");

array = new int[2097152];

prog.FillArray(array, 2097152);

timer.Reset();

timer.Start();

for (int i = 0; i < 10000000; i++)

temp = prog.binary\_search(array, -1, 2097152);

timer.Stop();

Console.WriteLine("Time8 : " + timer.ElapsedMilliseconds + " ms");

Console.ReadKey();

}

int binary\_search(int[] array\_search, int target, int arraySize)

{

int first, mid, last;

first = 0;

last = arraySize - 1;

while (first <= last)

{

mid = (first + last) / 2;

if (array\_search[mid] > target)

last = mid - 1;

else if (array\_search[mid] < target)

first = mid + 1;

else

return mid;

}

return -1;

}

public void FillArray(int[] array, int n)

{

for (int i = 0; i < n; i++)

{

array[i] = i;

}

}

}

}