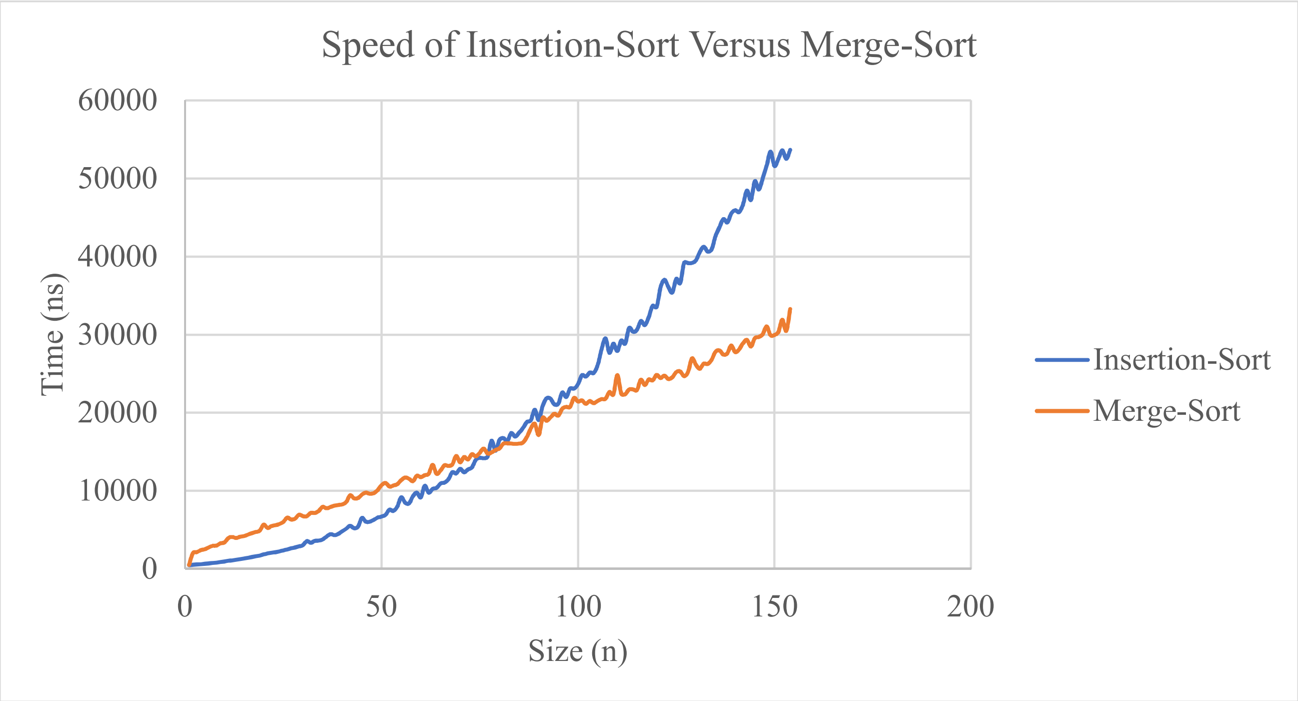
```
// Merge-Sort Algorithm
// By Anna DeVries

#include <iostream>
#include <cstdlib>
#include <string.h>
#include <chrono>

// Sorts vector
void merge(int v[], int left, int midpoint, int right){
    int n_1 = midpoint - left + 1;
    int n_2 = right - midpoint;

    int* L = new int[n_1];
    int* R = new int[n_2];

    for (int i = 0; i < n_1; i++){
        L[i] = v[left + i];
    }

    for (int j = 0; j < n_2; j++){
        R[j] = v[midpoint + 1 + j];
    }

    int i = 0;
    int j = 0;
    int k = left;
    while(i < n_1 && j < n_2){
        if(L[i] <= R[j]){
            v[k] = L[i];
            i++;
        }
        else{
            v[k] = R[j];
            j++;
        }
        k++;
    }

    while(i < n_1){
        v[k] = L[i];
        i++;
        k++;
    }

    while(j < n_2){
        v[k] = R[j];
        j++;
        k++;
    }

    delete[] L, R;
}

void merge_sort(int v[], int left, int right){
    int midpoint;

    if (left < right){
        midpoint = left + (right - left) / 2;
        merge_sort(v, left, midpoint);
        merge_sort(v, midpoint + 1, right);
        merge(v, left, midpoint, right);
    }
}

// Prints vector
void print_vector(int v[], int n){
    int i;
    std::cout << "Vector: ";
    for (i = 0;i < n; i++)
        std::cout << " " << v[i];
    std::cout << std::endl;
}

int main(int argc, char* argv[]){
    // Check that there are two arguments
    if( argc != 2){
        std::cout << "Usage: " << argv[0] << " <size>\n";
        return EXIT_FAILURE;
    }

    // Check that argv[1] is a valid integer
    char* arg = argv[1];
    for(int i = 0; i < strlen(arg); i++) {
        if(arg[i] < '0' || arg[i] > '9'){
            std::cout << "Please enter an integer \n";
            std::cout << "Usage: " << argv[0] << " <size>\n";
            return EXIT_FAILURE;
        }
    }

    // Converts user input n to an integer
    int n = atoi(argv[1]);
    std::cout << "Size n = " << n << std::endl;

    // Allocates space for array and fills in array at size n as worst-case scenario
    int numbers = n - 1;
    int* v = new int[n];
    for(int i = 0; i < n; i++){
        v[i] = numbers;
        numbers--;
    }

    //print_vector(v, n);

    // Initialize clock
    auto start = std::chrono::high_resolution_clock::now();

    //print_vector(v,n);

    // Performs sorting operations
    merge_sort(v, 0, n-1);

    auto finish = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = finish - start;
    std::cout << "Duration to sort (sec): " <<
    std::chrono::duration_cast<std::chrono::nanoseconds>(finish - start).count() << "
    ns\n";

    //print_vector(v, n);

    // Frees memory
    delete[] v;
```

```cpp
// Insertion Sort Algorithm
// By Anna DeVries

#include <iostream>
#include <cstdlib>
#include <string.h>
#include <chrono>

// Sorts vector
void insertion_sort(int v[], int n){
    int value;
    int i, j;

    for (i=1;i<n;i++){
        value = v[i];
        j = i-1;
        while(j>=0&&v[j]>value){
            v[j+1] = v[j];
            j--;
        }
        v[j+1]=value;
    }
}

// Prints vector
void print_vector(int v[], int n){
    int i;
    std::cout << "Vector: ";
    for (i=0;i<n;i++)
        std::cout << " " << v[i];
    std::cout << std::endl;
}

int main(int argc, char* argv[]){
    // Check that there are two arguments
    if( argc != 2){
        std::cout << "Usage: " << argv[0] << " <size>\n";
        return EXIT_FAILURE;
    }

    // Check that argv[1] is a valid integer
    char* arg = argv[1];
    for(int i = 0; i < strlen(arg); i++) {
        if(arg[i] < '0' || arg[i] > '9'){
            std::cout << "Please enter an integer \n";
            std::cout << "Usage: " << argv[0] << " <size>\n";
            return EXIT_FAILURE;
        }
    }

    // Converts user input n to an integer
    int n = atoi(argv[1]);
    std::cout << "Size n = " << n << std::endl;

    // Allocates space for array and fills in array at size n as worst-case scenario
    int numbers = n - 1;
    int* v = new int[n];
    for(int i = 0; i < n; i++){
        v[i] = numbers;
        numbers--;
    }

    // Initialize clock
    auto start = std::chrono::high_resolution_clock::now();

    //print_vector(v,n);

    // Performs sorting operations
    insertion_sort(v,n);

    auto finish = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = finish - start;
    std::cout << "Duration to sort (sec): " << std::chrono::duration_cast<std::chrono::nanoseconds>(finish - start).count() << " ns\n";

    //print_vector(v,n);

    // Frees memory
    delete[] v;
}
```

```
## Bash Script For Comparing Algorithms
## By Anna DeVries

#!/bin/bash

g++ -std=c++11 insertion_sort.cc -o insertion_sort
g++ -std=c++11 merge_sort.cc -o merge_sort

file=output.csv
func1=./insertion_sort
func2=./merge_sort
n=0

i=`$func1 $n | sed -n '2 p' | awk '{print $5}'`
j=`$func2 $n | sed -n '2 p' | awk '{print $5}'`

#while [ $(echo "$i >= $j" |bc --mathlib) -eq 1 ]; do
while [ $j -ge $i ]; do
    n=$((n+1))
    i=`$func1 $n | sed -n '2 p' | awk '{print $5}'`
    j=`$func2 $n | sed -n '2 p' | awk '{print $5}'`
    #echo n: $n
    #echo Insertion_Sort: $i
    #echo Merge-Sort:     $j
    echo "$n,$i,$j" >> $file
done

count=0
while [ $count -lt 100 ]; do
    n=$((n+1))
    i=`$func1 $n | sed -n '2 p' | awk '{print $5}'`
    j=`$func2 $n | sed -n '2 p' | awk '{print $5}'`
    echo "$n,$i,$j" >> $file
    count=$((count+1))
done
```

I ran each algorithm 10 times and averaged the time at size n across the 10 iterations, results are below. On average, I found that merge-sort begins to perform quicker than insertion-sort at size n 78. Insertion-sort took 16419 ns while merge-sort took 14915 ns at size 78.



Average Time in 10 Iterations at each Size n

Size (n)	Insertion-Sort (ns)	Merge-Sort (ns)
1	489	522
2	544	2036
3	585	2148
4	606	2414
5	653	2522
6	694	2756
7	756	2957
8	812	2992
9	875	3288
10	953	3384
11	1042.3	3956.4
12	1095	4090.9
13	1184.6	3962.3
14	1264	4120.5
15	1345.1	4220.6
16	1430.7	4392.7
17	1516.2	4574.6
18	1620.4	4722.1
19	1701.8	4908.1
20	1867.8	5688.6
21	1978.6	5248.1
22	2090.2	5489
23	2146.9	5604.6
24	2239.3	5747.2
25	2377.3	6020.5
26	2495.7	6570.1
27	2643.6	6324.5
28	2748.4	6435.2
29	2884.1	6946.2
30	3055.5	6756.9
31	3563.8	6756.5
32	3346.4	7187.8
33	3596.4	7159.2
34	3626.5	7424.4
35	3784.8	7947.4
36	4139.5	7775
37	4453.2	7927
38	4325.4	8086.6
39	4486.9	8194.8
40	4838.9	8266.6
41	5149.3	8562.1
42	5527.7	9434.5
43	5211.2	9037.6

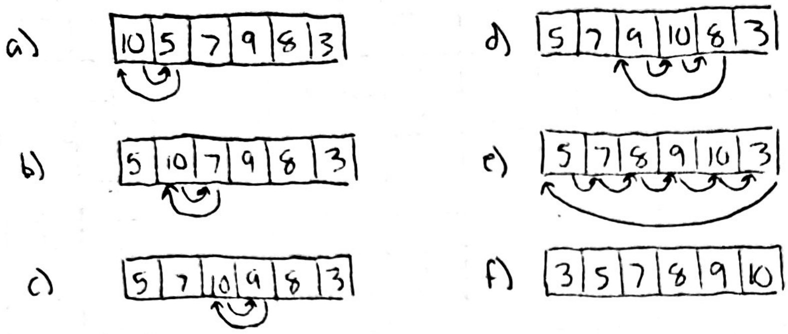
44	5411.4	9074.4
45	6530.7	9520.7
46	6042.3	9773.3
47	6048.8	9630.2
48	6280.1	9703
49	6569	10082.7
50	6701.4	10728
51	6926.3	11014.5
52	7590.9	10549.5
53	7428.8	10688.5
54	7975.8	10851.7
55	9174.5	11368.7
56	8486.1	11701.8
57	8394	11537.4
58	9327.7	11242.1
59	9761.4	11941.9
60	9166.5	11742.9
61	10647.4	12002.4
62	9771.4	12190.2
63	10248.1	13331.5
64	10382.8	12179.7
65	10936.8	12595.4
66	11060.3	13269.4
67	11474.2	13192.8
68	12377.9	13380.4
69	12231.2	14466
70	12815.5	13652.8
71	12374.6	14337.1
72	12743.8	14023.3
73	13005.5	14690.7
74	13981.2	14411.7
75	14224.3	14876.5
76	14174.2	15416.4
77	14386.3	14778
78	16419.7	14915.1
79	15138.1	15230.1
80	16513.8	15429.2
81	16766	16046.9
82	16395.6	16084.7
83	17410.6	16054.4
84	16946.9	16008.9
85	17440.3	16054.1
86	18040.6	16189.4
87	18821.7	16908.1
88	19035.6	18007.2
89	20392.5	18596.6
90	19069.6	17150.1
91	20887.7	19342.6
92	21848.6	18973.3
93	21801	19422.1
94	21069.2	19873.5
95	21178.7	19645
96	22597.1	20522.3
97	22022.7	20745.7
98	23092.6	20759.8
99	23094	21906.9
100	23655	21425.9
101	24803	21589.2
102	24625.9	21139.4
103	25151.3	21497.8
104	25098.6	21225.2
105	26102.2	21522.8
106	28123	21746.5
107	29518.5	21794.4
108	27674	22666
109	28854	22336.9
110	27903.8	24830
111	29251.4	22445.9
112	28865.7	22363.5
113	30848.4	22981.7
114	30369.3	22991.9
115	30614.9	22902.1
116	31750.3	24223
117	31226.5	23551.9
118	32239.5	24263.3
119	33706.7	24147.5
120	33538.4	24843.2
121	36047.8	24450.1
122	37031.2	24756.4
123	36093	24315.8
124	35398.5	24552.5
125	37151.2	25208.9
126	36612.1	25302.9
127	39191.9	24685.3
128	39161.3	25208.3
129	39184.7	26958.1
130	39514.1	26150.7
131	40573.8	25628.3
132	41256.55556	26297.22222
133	40626.44444	26247.11111

134	40924.77778	26779.11111
135	42639.66667	27821.22222
136	43753.33333	27975.33333
137	44799.88889	27448
138	44367.33333	27603.33333
139	45516.44444	28614.33333
140	45932.22222	27769.33333
141	45697	28121
142	46679.33333	28899.77778
143	48459	29330.22222
144	47239.33333	28476.88889
145	49652.33333	29546.77778
146	48576.66667	29703.88889
147	50010.88889	30042.11111
148	51602.55556	31052.22222
149	53438	29932.88889
150	51604.77778	29966.22222
151	52497.66667	30358
152	53606.66667	31913.77778
153	52511.625	30480.625
154	53659.75	33280.25

Anna DeVries | Homework #1

Question 2

Insertion Sort



a) 10 | 5 | 7 | 9 | 8 | 3

b) 5 | 10 | 7 | 9 | 8 | 3

c) 5 | 7 | 10 | 9 | 8 | 3

d) 5 | 7 | 9 | 10 | 8 | 3

e) 5 | 7 | 8 | 9 | 10 | 3

f) 3 | 5 | 7 | 8 | 9 | 10

Partition Subroutine:

First Partition	Second Partitions
a) |10 5 7 9 8 ③ pivot point	g) |3 10 5 7 9 ⑧ pivot point
b) |10|5 7 9 8 ③	h) |3|10|5 7 9 ⑧
c) |10 5|7 9 8 ③	i) 3 5|10|7 9 ⑧
d) |10 5 7|9 8 ③	j) 3 5 7|10|9 ⑧
e) |10 5 7 9|8 ③	k) 3 5 7|10 9|⑧
f) |10 5 7 9 8|③	l) 3 5 7 ⑧ 10 9
g) ③ 10 5 7 9 8	and so on

each one partition compares the array values to the "pivot point" such that at the end $\boxed{\leq x \quad (x) \quad \geq x}$

Above shows two rounds of portions.

Anna DeVries | Homework #1

Question 3 True or False

$n+3 \in \Omega(n)$

assume $c=1$ $n>1$

Then **True**

$\Omega(g(n)) = f(n)$

$0 \leq c\, g(n) \leq f(n)$ for $n \geq n_0$

$c \cdot n \leq n+3$

$c \leq 1 + \frac{3}{n}$

$n+3 \in O(n^2)$

assume $c=1$ $n \geq 3$

Then **True**

$O(g(n)) = f(n)$

$0 \leq f(n) \leq c\,g(n)$ for $n \geq n_0$

$n+3 \leq c n^2$

$1 + \frac{3}{n} \leq cn$

$n+3 \in \Theta(n^2)$

False

$\Theta(g(n)) = f(n)$ for $n \geq n_0$

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ $n \geq n_0$

$c_1 n^2 \leq n+3 \leq c_2 n^2$

$n+3 \in O(n^2)$ and $n+3 \in \Omega n^2$

$n+3 \leq c \cdot n^2$ | $cn^2 \leq n+3$

$1 + \frac{3}{n} \leq cn$ | $cn \leq 1 + \frac{3}{n}$

assume $c=1$ $n \geq 3$

then this is true but | as $n \to \infty$, $cn \cancel{\leq 1 + \frac{3}{n}}$ False

$2^{n+1} \in O(n+1)$

False

$O(g(n)) = f(n)$

$0 \leq f(n) \leq c\,g(n)$ for $n \geq n_0$

$2^{n+1} \leq c(n+1)$

$2^{1+1} \leq 1(1+1)$ X

$2^{n+1} \in \Theta(2^n)$

False

$\Theta(g(n)) = f(n)$ for $n \geq n_0$

$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$

and

$2^{n+1} \in O(2^n)$ | $2^{n+1} \in \Omega(2^n)$

$2^{n+1} \leq c \cdot 2^n$ | $c \cdot 2^n \leq 2^{n+1}$

False | True

Question 4

Using the master method, determine T(n) for the following:

$T(n) = 8T\left(\frac{n}{2}\right) + n$

$a=8 \quad b=2 \quad f(n)=n$

$n^{\log_b a} = n^{\log_2 8} = n^3$

$f(n) = O(n^{3-\epsilon})$ since n^3 is polynomically larger than $f(n)$ for $\epsilon = 1$ case 1 applies

$\underline{\underline{T(n) = \Theta(n^3)}}$

$T(n) = 8T\left(\frac{n}{2}\right) + n^2$

$a=8 \quad b=2 \quad f(n)=n^2$

$\underline{\underline{T(n) = \Theta(n^3)}}$

$n^{\log_b a} = n^3$

$f(n) = O(n^{3-\epsilon})$ since n^3 is larger than $f(n)$ for $\epsilon = 1$ case 1 applies

$T(n) = 8T\left(\frac{n}{2}\right) + n^3$

$a=8 \quad b=2 \quad f(n)=n^3$

$\underline{\underline{T(n) = \Theta(n^3 \lg n)}}$

$n^{\log_b a} = n^3$

$f(n) = \Theta(n^3)$ since $n^{\log_8 2} = n^3$ and $f(n) = n^3$ case 2 applies

$T(n) = 8T\left(\frac{n}{2}\right) + n^4$

$\underline{\underline{T(n) = \Theta(n^4)}}$

$n^{\log_b a} = n^3$

$n^4 = \Omega(n^{3+\epsilon})$ for $\epsilon > 0$ since $n^3 < f(n)$ and

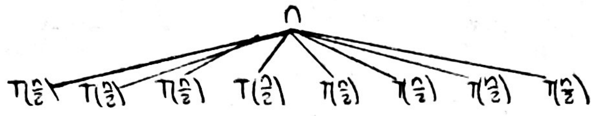
$\frac{a}{b^k} = \frac{7}{3^4} = 0.0864 < 1$ so case 3 applies

Ann DeVries | Homework #1

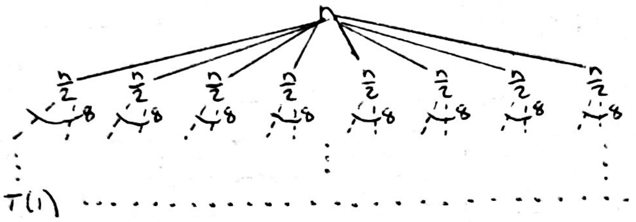
Question 5 (extra)

Draw a recursion tree for $T(n) = 8T(\frac{n}{2}) + n$. And prove $T(n)$ by substitution method.

$T(n) =$



$T(n) =$



n

$\frac{8}{2}n = 4n$

$4^k n = 4^{\log n}$

$\Theta(n^3) + \sum_{k=0}^{\lg n - 1} 4^k n$

↘ geometric series

$\frac{4^{k+1} - 1}{4 - 1}$

depth $= \log_2 n = \lg n$

width $= \Theta(n^{\log_2 8}) = \Theta(n^3)$

Since the geometric series is dominated by its final term $(4)^L n$ and each leaf contributes to this (n^3 leaves),

$$T(n) = \Theta(n^3)$$

$0 \leq c_1 n^3 \leq T(n)$

$8(c(n/2)^3) + n \leq T(n)$

$cn^3 + n \leq T(n)$

$0 \leq T(n) \leq c_2 n^3$

$T(n) \leq 8(c(n/2)^3) + n$

$T(n) \leq \cancel{8}c\left(\frac{n^3}{\cancel{2^3}}\right) + n$

$T(n) \leq cn^3 + n$

~~assume T(1)=1~~

at $k=1$ $T(n) = 8T(n/2) + n$

$k=2$ $T(n) = 16T(n/4) + n/2$

$k=3$ $T(n) = 32T(n/8) + n/4$

and so on such that $T((\frac{n}{2^k}) = 1) = C$ $T(1) = c$

$k = \lg n$ $n = 2^k$

$T(n) = \frac{4^{k+1} - 1}{4 - 1}$ $\rightarrow T(n) = \Theta(n^3)$