**Part 1: Solve the following recurrences using substitution method.**

We can use the substitution method to establish either upper or lower bounds on a recurrence equation.

1. **T(n)= T(n-3) + 3 lg n.**

**Our guess: T(n)= O(n lg n)**

**Prove T(n) <= cn lg n for c > 0**

**For n=1**,

T(1) = T(1-3) + 3 lg 1

= -2 + 3(0)

= -2

**Inductive step:**

Upper Bound T(n) < =cn lg n for c>0

T(n)= T(n-3) + 3 lg n.

<= (cn lg n – 3) + 3 lg n

<= lg n ((cn – 3) +3))

<= cn lg n (for c >0)

**Therefore T(n) = O (n log n)**

1. **T(n)=4T(n/3) + n**

**Our guess: T(n)= O(n ^ lg3 4)**

**Prove T(n) <= cn^log3 4 for c > 0**

**For n=1,**

T(1)=> 4 ((n ^ log3 4)/3) + n

=> n(4/3 n^log3 4 +1)

This proves T(n) is not <= cn^log3 4

**Improved guess:**

Upper Bound T(n) <= cn^log3 4 - 3n c>0

**Inductive step:**

T(n)= 4T(n/3) + n

<= 4(c (n/3)^log3 4 – (3n/3)) +n

<= 4c (n/3)^log3 4 - 4n +n

<= 4c (n/3)^log3 4 - 3n

**Therefore T(n) = O (n ^ log3 4 - 3n)**

1. **T(n)=T(n/2) +T(n/4) + T(n/8) +n**

**Our guess: T(n)= O(n)**

**Prove T(n) <= cn for c > 0**

**For n=1,**

T(1) =T(n/2) +T(n/4) + T(n/8) +n

= 1/2 +1/4 +1/8 +1

= 23/8

**Inductive step:**

Upper Bound T(n) <= dn

T(n)= T(n/2) +T(n/4) + T(n/8) +cn

<= dn/2 +dn/4 +dn/8 +cn

<= dn(7/8) +cn

<= n (7d/8 + c)

d(7/8) <= 0 which is therefore c >= +d ⅞

**T(n)=O(n)**

Lower Bound T(n) >= dn

T(n)= T(n/2) +T(n/4) + T(n/8) +cn

>= dn/2 +dn/4 +dn/8 +cn

>= dn(7/8) +cn

>= n (7d/8 +c)

d(7/8) +c >= 0

**Therefore T(n)=Ω(n)**

1. **T(n)=4T(n/2)+n^2**

**Our guess: T(n)= O(n^2)**

**Prove T(n)<=cn^2 for c > 0**

**For n=1,**

T(1) = 4T(n/2) + n^2

= 4(1/2) + 1^2

= 3

**Inductive step:**

T(n)= 4T(n/2)+n^2

<= 4 c (n/2)^2 + n^2

<= c n^2 + n^2

<= n^2 (c+1)

This proves T(n) is not <= cn^2

**Improved guess**

T(n) < =cn^2 - n , c>0

T(n) = 4T(n/2) + n

<= 4(c(n/2)^2 – (n/2)) + n

<= cn^2 – 2n + n

<= cn^2 – n

**Therefore T(n) = O(n^2)**

**Part 2: Radix sort on strings**

**1. Modified insertion sort algorithm**

The function “radix\_sort” uses insertion sort algorithm to sort string. The “insertion\_sort” function is the improvised version and insertion\_sort\_ori is the original function.

void radix\_sort(char\*\* A, int l, int r, int\* A\_len)

{

int k = 0;

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

{

if(A\_len[i] > k)

{

k = A\_len[i];

}

}

for(int i=k-1;i>=0;i--)

{

int d=i;

insertion\_sort(A,l,r,d,A\_len);

}

}

void insertion\_sort(char\*\* A, int l, int r, int d, int\* A\_len)

{

int i;

char\* key;

int temp\_len;

for (int j = l+1; j <= r; j++)

{

key = A[j];

temp\_len=A\_len[j];

i = j - 1;

int ascii1=0;

int ascii2=0;

if(d < A\_len[i])

ascii1= (int)(A[i][d]);

if(d < A\_len[j])

ascii2= (int)key[d];

while ((i >= l) && (ascii1 > ascii2))

{

A[i+1] = A[i];

A\_len[i+1]=A\_len[i];

i = i - 1;

ascii1 = 0;

if(d < A\_len[i] && i >= l)

ascii1= (int)(A[i][d]);

}

A[i+1] = key;

A\_len[i+1]=temp\_len;

}

}

**3. Counting sort algorithm for strings.**

void radix\_sort\_count(char\*\* A,char\*\* D, int l, int r,int\* A\_len)

{

int max = 0;

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

{

if(A\_len[i] > max)

{

max = A\_len[i];

}

}

for(int i=max-1;i>=0;i--)

{

int d=i;

int k=256;

counting\_sort(A,D,k,r,d,A\_len);

}

D = A;

}

void counting\_sort(char\*\* A, char\*\* B, int k, int n, int d, int\* A\_len)

{

int c[k];

int newLen[n+1];

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

{

c[i]=0;

}

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

{

int asc=48;

if(d<A\_len[j])

asc=int(A[j][d]);

c[asc]=c[asc]+1;

}

for(int i=1;i<=k;i++)

{

c[i]=c[i]+c[i-1];

}

for(int j = n;j >= 0;j--)

{

int asc=48;

if(d < A\_len[j])

asc=(int)(A[j][d]);

B[c[asc]-1] = A[j];

newLen[c[asc]-1]=A\_len[j];

c[asc] = c[asc] - 1;

}

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

{

A[c] = B[c];

A\_len[c]=newLen[c];

}

}

**2. Modified insertion sort algorithm. Measure runtime performance.**

**4. Radix sort algorithm. Measure runtime performance.**

The results for Question 2 and 4 are depicted in the below table and line chart which shows the variation of time for various inputs.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input m** | **Input n** | **Q2** | | **Q4** | |
| **Random Generator(ms)** | **Radix using insertion sort(ms)** | **Random Generator(ms)** | **Counting sort(ms)** |
| 25 | 10000 | 2 | 13 | 1 | 22 |
| 50 | 10000 | 4 | 38 | 3 | 48 |
| 75 | 10000 | 6 | 63 | 4 | 27 |
| 25 | 25000 | 3 | 22 | 2 | 48 |
| 50 | 25000 | 11 | 83 | 4 | 55 |
| 75 | 25000 | 20 | 148 | 7 | 72 |
| 25 | 50000 | 7 | 50 | 3 | 89 |
| 50 | 50000 | 29 | 111 | 8 | 116 |
| 75 | 50000 | 26 | 211 | 12 | 132 |
| 25 | 75000 | 10 | 62 | 6 | 139 |
| 50 | 75000 | 26 | 159 | 14 | 161 |
| 75 | 75000 | 38 | 318 | 17 | 215 |
| 25 | 100000 | 15 | 101 | 9 | 181 |
| 50 | 100000 | 35 | 224 | 16 | 231 |
| 75 | 100000 | 43 | 347 | 24 | 269 |