EC 504 Data Structure
HW1

Name: Yangruirui Zhou **BU ID： U16997747**

1. a)

3-2 Relative asymptotic growths

Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that $k \geq 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with “yes” or “no” written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^ϵ					
b.	n^k	c^n					
c.	$\sqrt{n}$	$n^{\sin n}$					
d.	2^n	$2^{n/2}$					
e.	$n^{\lg c}$	$c^{\lg n}$					
f.	$\lg(n!)$	$\lg(n^n)$					

	O	o	Ω	ω	Θ
a	y	y	n	n	n
b	y	y	n	n	n
c	n	n	n	n	n
d	n	n	y	y	n
e	y	n	y	n	y
f	y	n	y	n	y

b)

$$\prod_{k=1}^{n}\left(1-\frac{1}{k^2}\right) < (1-\frac{1}{n})^n = 1 = n^{\frac{1}{n}} < \sum_{k=1}^{n}\frac{1}{k} < ln(ln(n)) < ln(n) < 3^{ln(n)} < n^{1+\cos n} < n^2 + n^{-2}$$

$$< n^2 + 3n + 5 < n^2 + 3nlog(n) + 5 < \sum_{k=1}^{log(n)}\frac{n^2}{2^k} < n! < (1+n)^n < n^{n^2-1} < n^{n^2} + n!$$

(c) Because $T(n) = c_1 n + c_2 n \log_2 n$, and $T(n) = 2T(n/2) + n$. We can get following equations:

$$c_1 n + c_2 n \log_2 n = 2 \times \left(c_1 \frac{n}{2} + c_2 \frac{n}{2} \log_2 \frac{n}{2}\right) + n = c_1 n + c_2 n \log_2 \frac{n}{2} + n$$

$$\because n \neq 0 \ \therefore c_2 \log_2 n = c_2 \log_2 \frac{n}{2} + 1$$

$$c_2 \log_2 n = c_2 \log_2 n - c_2 + 1$$

$$\therefore c_2 = 1,\ c_1 = T(1)$$

(extra) From $T(n) = c_1 n^\gamma + c_2 n^k,\ T(n) = aT(n/b) + n^k$

$$c_1 n^\gamma + c_2 n^k = a[c_1 (\frac{n}{b})^\gamma + c_2 (\frac{n}{b})^k] + n^k = ac_1 \frac{n^\gamma}{b^\gamma} + ac_2 \frac{n^k}{b^k} + n^k$$

$$\because\ b^\gamma = a, \gamma = k\ \therefore c_1 n^\gamma + c_2 n^k = \frac{ac_1}{b^\gamma} n^\gamma + \frac{ac_2}{b^k} n^k + n^k = c_1 n^\gamma + c_2 b^{\gamma - k} n^k + n^k$$

$$\therefore (c_1 + c_2) n^\gamma = (c_1 + c_2) n^\gamma + n^\gamma \therefore n^\gamma = 0$$

Thus, it means only when n=0, the equation can be satisfied.

From $T(n) = c_1 n^\gamma + c_2 n^\gamma \log_2 n,\ T(n) = aT(n/b) + n^\gamma$

$$c_1 n^\gamma + c_2 n^\gamma \log_2 n = a[c_1 (\frac{n}{b})^\gamma + c_2 (\frac{n}{b})^\gamma \log_2 \frac{n}{b}] + n^\gamma = ac_1 \frac{n^\gamma}{b^\gamma} + ac_2 \frac{n^\gamma}{b^\gamma} (\log_2 n - \log_2 b) + n^\gamma$$

$$c_1 n^\gamma + c_2 n^\gamma \log_2 n = \frac{ac_1}{b^\gamma} n^\gamma + \frac{ac_2}{b^\gamma} n^\gamma (\log_2 n - \log_2 b) + n^\gamma = c_1 n^\gamma + c_2 n^\gamma (\log_2 n - \log_2 b) + n^\gamma$$

$$0 = c_2 n^\gamma (-\log_2 b) + n^\gamma$$

$$\because n \neq 0\ \therefore c_2 = \log_b 2,\ c_1 = T(1)$$

2.

(a) In each recursion, we need to execute the conditional statement to judge if n equals to 0 and complete the multiple operation A(n-1) * A(n-1), so totally 2 executions. And in the last recursion, we will only execute 2 statements: first, is the conditional statement and the second one is “return 1”. Thus, we can have following equations:

$$T(0) = 2$$

$$T(n) = 2T(n-1) + 2 = 2\,(2\mathrm{T}(\mathrm{n}-2) + 2) + 2 = 2^{\mathrm{k}}\,\mathrm{T}(\mathrm{n}-\mathrm{k}) + 2^{\mathrm{k}+1} - 2$$

From the structure, we can assume $T(n) = c_1 2^n + c_2$ and substitute it into $T(n) = 2T(n-1) + 2$

$$c_1 2^n + c_2 = 2(c_1 2^{n-1} + c_2) + 2 = c_1 2^n + 2c_2 + 2$$

$$c_2 = -2$$

Thus, we can substitute $c_2 = -2$ and $T(n) = c_1 2^n + c_2$ into $T(0) = 2$, and we can get $c_1 = 4$

$$T(n) = 2^{n+2} - 2 = \Theta(2^n)$$

(b) In each recursion (n>1), we need to execute the conditional statement to judge if n equals to 0 and if $\mathrm{B}(\mathrm{n}/2) \geq 10$ and execute one return statement, so totally 3 normal execution + twice B(n/2) call. When n=1, we will only need to call B(n/2) once, because B(1/2) = B(0) = 1 < 10. Thus, we totally need 3 normal execution + once B(n/2) call. When n=0, we will only need two execution includes judging if n equals to 0 and return 1, so T(0)=2. Thus, we can have following equations:

$$T(n) = \begin{cases} 2T(n/2) + 3 & n > 1, n \bmod 2 = 0 \\ 2T((n-1)/2) + 3 & n > 1, n \bmod 2 = 1 \\ T(0) + 3 = 5 & n = 1 \\ 2 & n = 0 \end{cases}$$

(i) When n>1 and n is even number:

$$T(n) = 2T(n/2) + 3 = 2\,(2\mathrm{T}(\mathrm{n}\,/\,4) + 3) + 3 = 2^{\mathrm{k}}\,\mathrm{T}(\mathrm{n}\,/\,2^{\mathrm{k}}) + 3 \times (2^{\mathrm{k}} - 1)$$

From the structure, we can assume $T(n) = c_1 n + c_2$

$$c_1 n + c_2 = 2 \times (c_1 \frac{n}{2} + c_2) + 3 = c_1 n + 2c_2 + 3 = c_1 n + 2c_2 + 3$$

$$\therefore c_2 = -3$$

Thus, we can substitute $c_2 = -3$ and $T(n) = 2T(n/2) + 3$ into $T(2) = 2T(1) + 3 = 13$

$$\therefore c_1 = 8$$

$$T(n) = 8n - 3$$

(ii) When n>1 and n is odd number:

$$T(n) = 2T((n-1)/2) + 3 = 2\,(2\mathrm{T}((\mathrm{n}-1)\,/\,4) + 3) + 3 = 2^{\mathrm{k}}\,\mathrm{T}((\mathrm{n}-1)\ /\,2^{\mathrm{k}}) + 3 \times (2^{\mathrm{k}} - 1)$$

From the structure, we can assume $T(n) = c_1(n-1) + c_2$

$$c_1 n + c_2 = 2 \times \left(c_1 \frac{n-1}{2} + c_2\right) + 3 = c_1 n - c_1 + 2c_2 + 3 = c_1 n - c_1 + 2c_2 + 3$$

$$\therefore c_1 - c_2 = 3$$

Thus, we can substitute $c_1 - c_2 = 3$ and $T(n) = c_1 n + c_2$ into $T(3) = 2T(1) + 3 = 13$

$$\therefore \begin{cases} c_1 = \frac{16}{3} \\ c_2 = \frac{7}{3} \end{cases}$$

$$T(n) = \frac{16}{3} n + \frac{7}{3}$$

So,

$$T(n) = \begin{cases} 8n - 3 & n > 1, n \bmod 2 = 0 \\ \frac{16}{3} n + \frac{7}{3} & n > 1, n \bmod 2 = 1 \\ 5 & n = 1 \\ 2 & n = 0 \end{cases} = \Theta(n)$$

(c) In each recursion (n>1), we need to execute the conditional statement to judge if n equals to 0 and if n equals to 1 and execute one return statement, so totally 3 normal execution + once D(n-1) call +twice D(n-2) call. And it is very easy to know the number of executions when n equals to 0 and 1. So we can get following equations:

$$T(n) = \begin{cases} T(n-1) + 2T(n-2) + 3 & n > 1 \\ 3 & n = 1 \\ 2 & n = 0 \end{cases}$$

From the structure, we can assume $T(n) = c_1 2^n + c_2$

$$c_1 2^n + c_2 = (c_1 2^{n-1} + c_2) + 2(c_1 2^{n-2} + c_2) + 3 = c_1 2^{n-1} + c_2 + 2c_1 2^{n-2} + 2c_2 + 3 = c_1 2^n + 3c_2 + 3$$

$$c_2 = -\frac{3}{2}$$

Thus, we can substitute $c_2 = -\frac{3}{2}$ and $T(n) = c_1 2^n + c_2$ into $T(2) = T(1) + 2T(0) + 3 = 10$ and T(3) we can find c_1 is not exist. So, the assumption fails, we have to analyze the borders of it.

(i) For the lower border, note that T(n) > $S(n) = S(n-1) + 2S(n-2)$ (n>1), which can be calculate by:
Let T(n)=r^n in the difference equation $S_{n+2} = S_{n+1} + 2S_n$ to obtain $r^2 - r - 2 = 0$, which yields the solutions $r \in \{2, -1\}$ and leads to the general form

$$S_n = A2^n + B(-1)^n$$

. Assume the values S_2=0 and S_3=1 then

$$S_n = \frac{2^{n-2} - (-1)^n}{3} = \Theta(2^n) \text{ and T(n) = } \Omega(2^n)$$

(ii) For the upper border, note that T(n) < H(n) $= 2H(n-1) + 2 = H(n-1) + 2H(n-2) + 4$ (n>1), which can be calculate by:
We can assume $H(n) = c_1 2^n + c_2$ and H_3=6

$$c_1 2^n + c_2 = 2(c_1 2^{n-1} + c_2) + 2 \therefore c_2 = -2, c_1 = 1$$

Thus, $H(n) = O(2^n)$

From above, we can know $\mathrm{T(n)} = \Theta(2^n)$

3. Pseudo-code description:

Get inputs from terminal, let nuts[] be the sets of nuts, bolts[] be the sets of bolts, n is their number. Swap() means value transition. PrintArray() means print two arrays on the console.

```
int find(char arr[], int low, int high, char midvalue, int flag)
// flag is used to remark the order of nut and bolt in TEST()
{
   int i = low;

   for(int j = low; j < high; j++)
   {
        if (flag > 0) a = arr[j]; b = midvalue;
             else a = midvalue; b = arr[j];

      // This is used to judge if midvalue is bigger than arr's element.
      // If it is true, I will put it to midvalue's left.
      if (TEST(a, b) * flag == -1)
      {
             Swap(arr[i],arr[j]);
```

```
            i++;
        }
        else if (TEST(a, b) == 0)
        {
                Swap(arr[j],arr[high]);
            j--;
        }
    }
    Swap(arr[i],arr[high]);
    return i;
}
```

// Function which works just like quick sort, low is the smallest subtitle of both arrays, high is the biggest subtitle of both arrays.

```
void Qsort(char nuts[], char bolts[], int low, int high)
{
    if (low < high)
    {
        int middle = find(nuts,low, high, bolts[high], 1)

        find(bolts, low, high, nuts[middle], -1);
        Qsort(nuts, bolts, low, middle - 1);
        Qsort(nuts, bolts, middle + 1, high);
     }
     PrintArray (nuts, bolts)
}
```

4. Coding work report

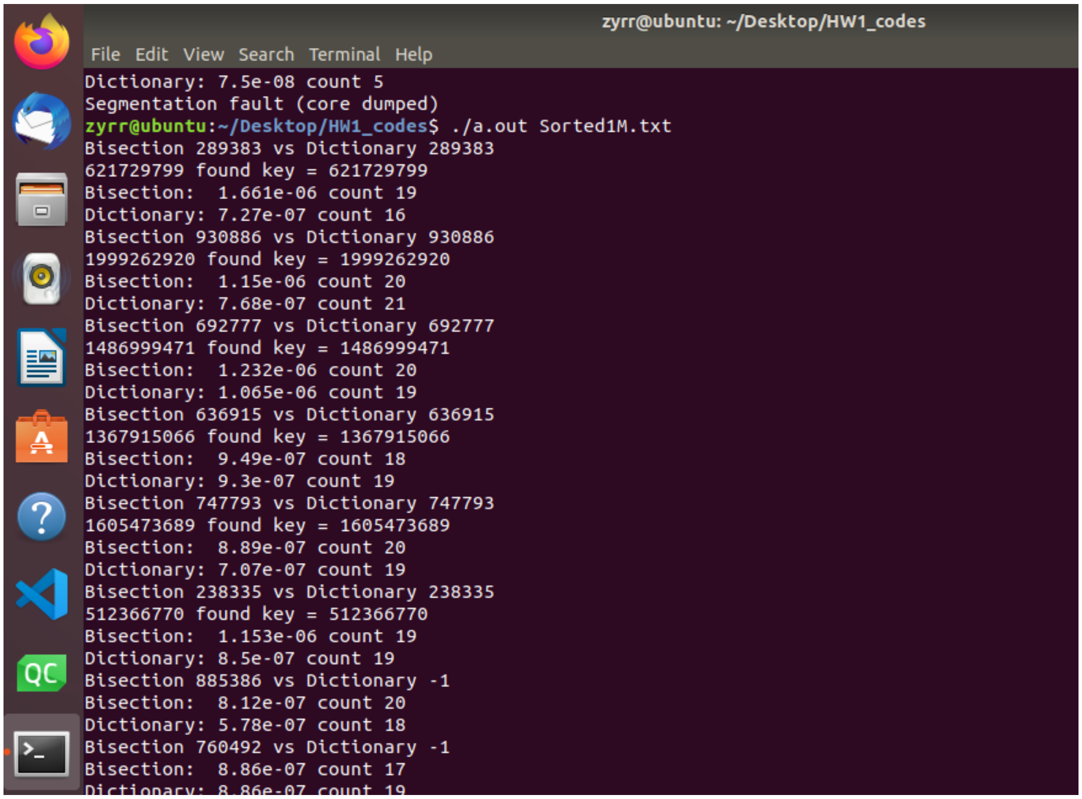

Fig1. Command line input and output

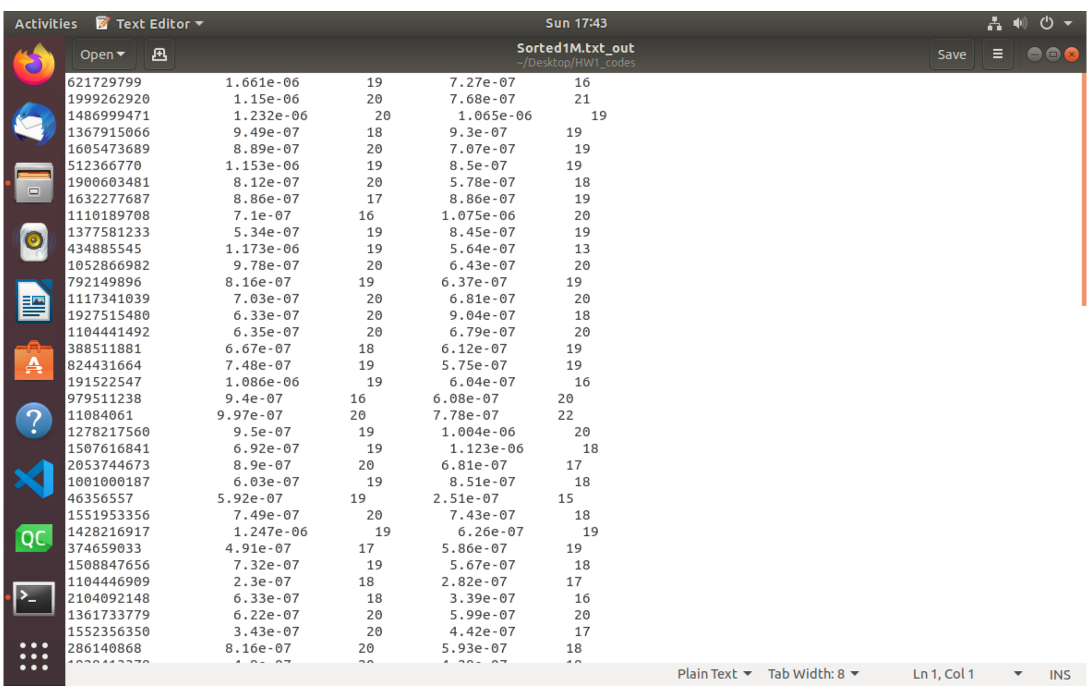

Fig2. Output file (Sorted1M.txt_out)

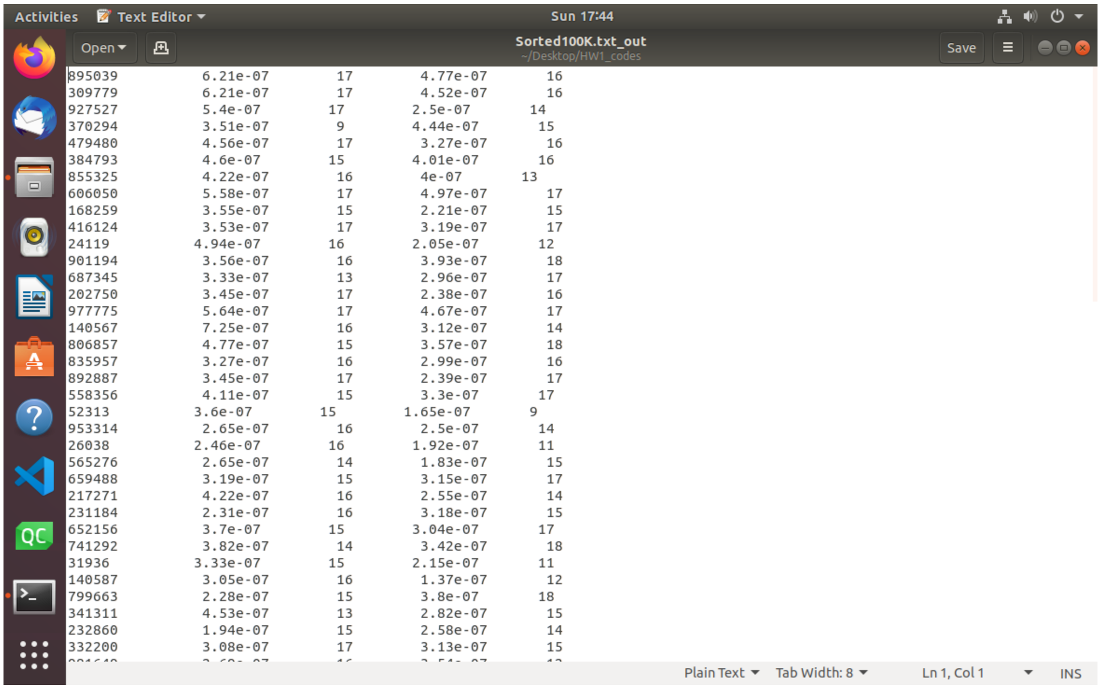

Fig3. Output file (Sorted100K.txt_out)

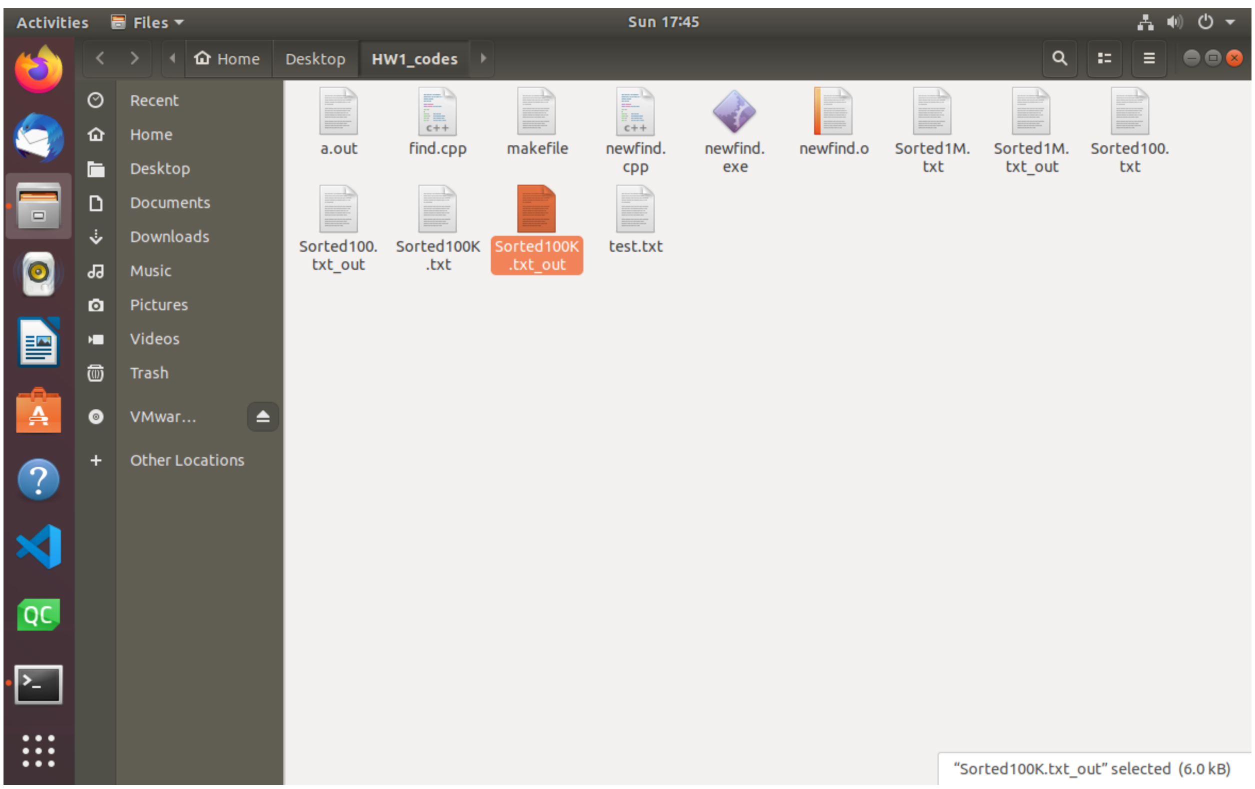

Fig4. All files I got in the HW1_codes folder