Agenda

i) Toggle string

ii) Sort an array of char

iii) longest palindromic substring

String

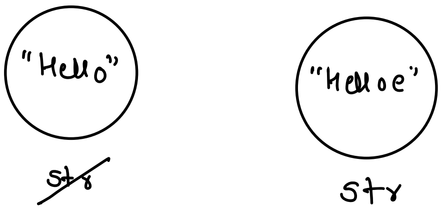
↳ seq of chars

char →	'a' - 'z'	(97 to 122)
	'A' - 'Z'	(65 to 91)
	'0' - '9'	(48 to 57)
	'@', '|', '#', ' '	(special char)

```
String str = "Hello";

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

        str += i;

}
```

$\sim O(n^2)$

```
String str = "Hello";
str += 'e';
```
→ O(n)



How to do concatenation related thing with better performance

→ StringBuilder

advantages of stringbuilder over strings

i) stringbuilder is mutable unlike strings
(also by using char[ ])

ii) concatenation becomes efficient

Q.1 Given a string, toggle every char.

str = a b c A e D

ans = A B C a E d

a	A	
97	65	⇒ 32

b	B	
98	66	⇒ 32

if char is UC → LC (ch+32)

else char is LC → UC (ch-32)

str = a D b
      0 1 2

i	ch	nch
0	'a' (LC → UC)	'A' (97-32 = 65)
1	'D' (UC → LC)	'd' (68+32 = 100)
2	'b' (LC → UC)	'B' (98-32 = 66)

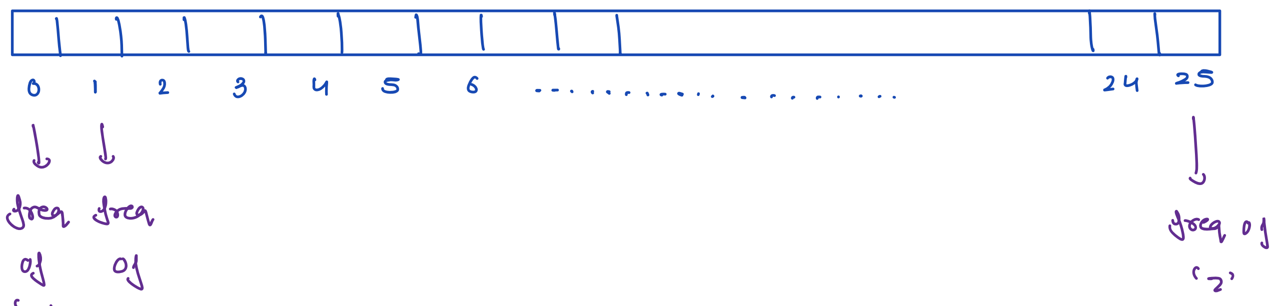
Q-2 Given a char [] , sort it lexicographically.
↳ (all chars are lowercase)

TC should be O(n)

A ⇒	a	d	a	b	c	b
	0	1	2	3	4	5
ans ⇒	a	a	b	b	c	d
	0	1	2	3	4	5

A ⇒	m	n	a	e	a	n
	0	1	2	3	4	5
ans ⇒	a	a	e	m	n	n
	0	1	2	3	4	5

Create freq array. (help us to calculate freq of every char)



A ⇒	a	d	a	b	c	b
	0	1	2	3	4	5

2	2	1	1				...		
0	1	2	3	4	5	6	....	24	25
a	b	c	d	e				y	z

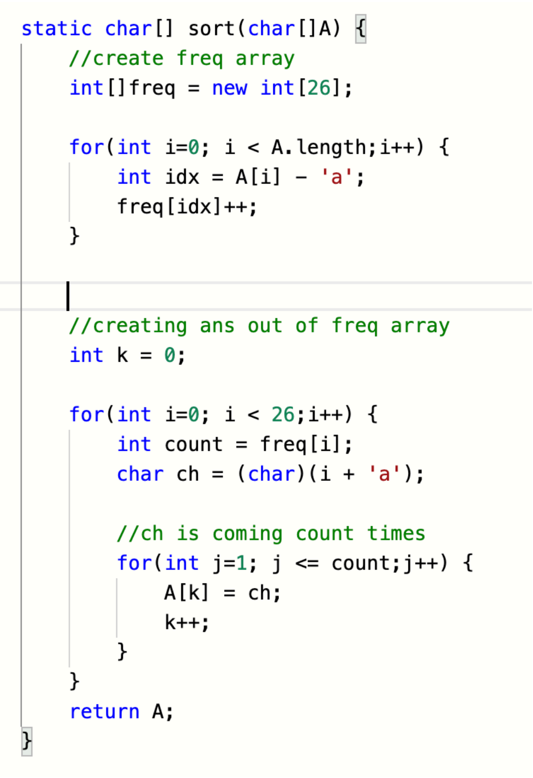
ans : aabbcc

idx = ch - 97

Steps :

i) create freq array

ii) create ans from freq array



```
static char[] sort(char[]A) {
    //create freq array
    int[]freq = new int[26];

    for(int i=0; i < A.length;i++) {
        int idx = A[i] - 'a';
        freq[idx]++;
    }


    //creating ans out of freq array
    int k = 0;

    for(int i=0; i < 26;i++) {
        int count = freq[i];
        char ch = (char)(i + 'a');

        //ch is coming count times
        for(int j=1; j <= count;j++) {
            A[k] = ch;
            k++;
        }
    }
    return A;
}
```

A = [a d c c a]

a	d	c	c	a
0	1	2	3	4

↙

freq =

2	0	2	1	0		0
0	1	2	3	4	. . . .	25
↓	↓	↓	↓	↓		↓
a	b	c	d	e		z

A =

a	a	c	c	d
0	1	2	3	4

K

i	count	ch
0	2	a
1	0	b
2	2	c
3	1	d

TC : O(n)

SC : O(1)

Substring

Subarrays for Arrays

substring for strings

String str = "Hello world";

H e l l o   w o r l d
0 1 2 3 4 5 6 7 8 9 10

what substring s = 3, e = 7 ⟹ "lo wo"

s = 2, e = 5 ⟹ "llo "

→ direct function to get substring of str from s to e.

str.substring(s, e+1);

↳ content from s to e

String str = "Hello world";

H e l l o   w o r l d
0 1 2 3 4 5 6 7 8 9 10

str.substring(2, 7) ⟹ "llo w"

H e [l l o   w] o r l d
0 1 2 3 4 5 6 7 8 9 10

str.substring(3, 9) ⟹ "lo wor"

H e l [l o   w o r] l d
0 1 2 3 4 5 6 7 8 9 10

Q-3 Longest palindromic substring

Given a string, find out the longest palindromic substring.

str = " a k m k d d k m p "

some palindromic substrings: kmk, dd, kddk,

mkddkm

ans: 6

Brute force: go on every substring, if it is palindromic it can be your ans.

```
int solve (String str) {

    int n= str.length();
    int ans= 0;
    for (int s=0; s<n ; s++) {

        for (int e=s; e<n; e++) {

            int len= e-s+1;
            if( isPal (str, s, e)==true) {

                ans= Math.max(ans, len);

            }

        }

    }
}
```

→ travelling on all substrings

todo

TC: $O(n^3)$

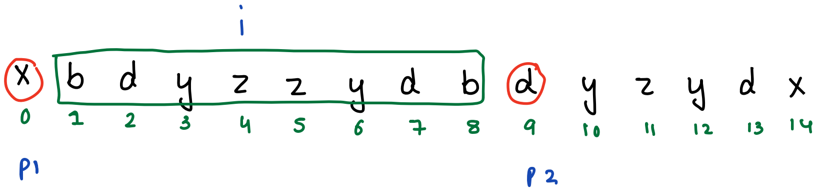
Expected TC : O(n^2)

	←	←	←	←	→	→	→	→	←	←		→	→	
x	b	d	y	z	z	y	d	b	d	y	z	y	d	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

even length substrings : zz, yzzy, dyzzyd,
bdyzzydb

odd length substring : z, yzy, dyzyd

```
int   LPS ( string str)   {

      int n= str.length();
      int ans= 1;

      // even length substrings                              TC : O(n²)

      for (int  i=0;  i<n-1;  i++)  {

            int p1 = i;

            int p2= i+1;

            ans= Math.max (ans, expand ( str, p1, p2));

      }
      // odd length substrings

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

            int p1 = i-1;

            int p2= i+1;

            ans= Math.max (ans, expand ( str, p1, p2));

      }
      return ans;
}
```



i = 4

$$\underbrace{p2 - p1 + 1}_{p1 \text{ to } p2} - 2 = \underbrace{p2 - p1 - 1}$$

```
int  expand ( string str, int p1, int p2) {

  while( p1 >= 0 && p2 < str.length() && str.charAt(p1)==str.charAt(p2)){
          p1--;
          p2++;
  }

   return p2-p1-1;
}
```

```
int  expand (string str, int p1, int p2) {
 while( p1 >= 0 && p2 < str.length() && str.charAt(p1)==str.charAt(p2))
          p1--;
          p2++;
   }
   return p2-p1-1;
}
```

str: a b c b a m m k
      0 1 2 3 4 5 6 7

ans = ~~1~~ 2

=> even length substrings

```
            i
a b c b a m m k
0 1 2 3 4 5 6 7
            p1 p
```

```
// even length substrings
for (int i=0; i<n-1; i++) {
      int p1 = i;
      int p2 = i+1;
      ans = Math.max (ans, expand (str, p1, p2));
}
```

odd length substrings

ans = ~~1~~ 5

```
a b c b a m m k
0 1 2 3 4 5 6 7
```

```
// odd length substrings
for (int i=1; i<n-1; i++) {
      int p1 = i-1;
      int p2 = i+1;
      ans = Math.max (ans, expand (str, p1, p2));
}
```

count total pairs

k = 12

1	5	5	5	8	7	7
0	1	2	3	4	5	6

$k - A[i]$

$12 - 7 = 5$

1 →	1
5 →	3
8 →	1
7 →	1

hashmap

count = 3 + 3

3 → 2 →

1	5	5	5	8	7	7
0	1	2	3	4	5	6

i
j

k = 12

ans= 6

1 2 3 3 5 5 7 7 10 10 10 14

k = 12

ans= 3 + 4