Agenda

i) Toggle string

ii) Sort an array of char

iii) longest palindromic substring

string → seq of char

char :	'a' - 'z'	[97 - 122]
	'A' - 'Z'	[65 - 91]
	'0' - '9'	[48 - 57]
	'#', '@', ' ', '.'	

```
string str = "Hello";
```

H	e	l	l	o
0	1	2	3	4

// can you change the i th char of str → NO

↳ strings are immutable

```
SOPln ( str.charAt(2) );      l
SOPln ( str.length() );       5
```

Q.1 Given a string, toggle chars and return ans string.

A = a B d K f m J

ans = A b D k F M j

a	A	b	B
97	65	98	66
⇒ 32		⇒ 32	

```
if (char is UC) {
    nch -> ch + 32
}
else {
    nch -> ch - 32
}
```

⇒ ch = 'A'
nch = 'A' + 32 = 65+32 = 97
= 'a'

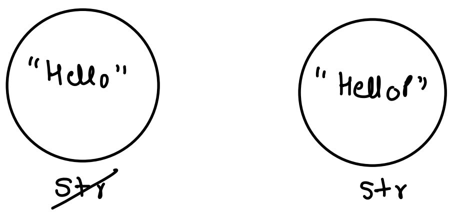
⇒ ch = 'b'
nch = 'b' - 32 = 98-32 = 66
= 'B'

// $\boxed{\text{doing concatenations in string is inefficient}}$

String str = "Hello";

str += 'p'; → O(n)

Java's String



Q.2 Given a char[] (lowercase chars), sort it inplace.

Expected TC: O(n)

A =	d	o	b	a	b	a
	0	1	2	3	4	5

Sort ↓

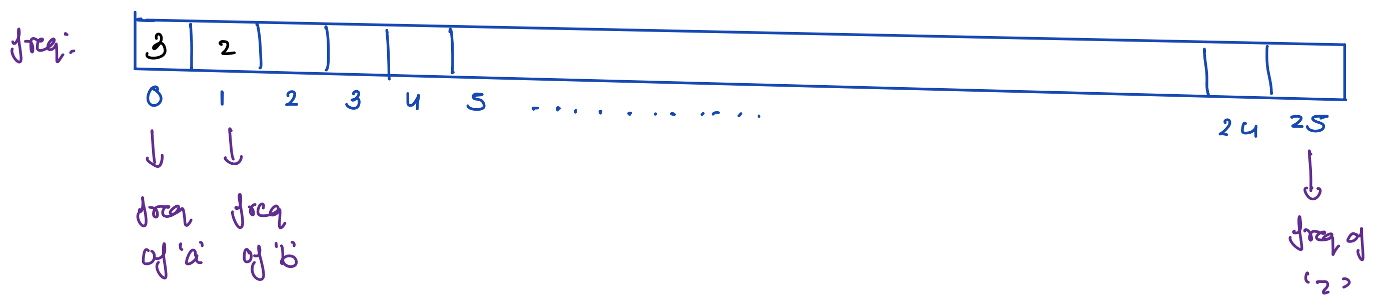
A =	a	a	b	b	d	o
	0	1	2	3	4	5

A =	e	f	p	a	f	e	b
	0	1	2	3	4	5	6

Sort ↓

A =	a	b	e	e	f	f	p
	0	1	2	3	4	5	6

A =	b	a	b	a	a
	0	1	2	3	4



i) create freq array of size 26 and fill it with the help of char[] given to you.

ii) with the help of freq array, build final answer.

→ How to create freq array ✓

char[] A = [b a c a c f c b]

char[] A =	b	a	c	a	c	f	c	b
	0	1	2	3	4	5	6	7

freq:

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

idx = ch - 'a'

ch	idx
'b'	'b' - 'a' = 1
'a'	'a' - 'a' = 0
'c'	'c' - 'a' = 2
'a'	'a' - 'a' = 0
⋮	

freq[idx]++;

```java
static void sort(char[]A) {
    //create freq array of size 26
    int[]freq = new int[26];
    for(int i=0; i < A.length;i++) {
        int idx = A[i] - 'a';
        freq[idx]++;
    }

    //build ans using the freq array
    int k=0;
    for(int i=0; i < 26;i++) {
        int count = freq[i];
        char ch = (char)(i + 'a');

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

A = [b a c a c f c b]

b	a	c	a	c	f	c	b
0	1	2	3	4	5	6	7

↙

freq =

2	2	3			1		. . . .	
0	1	2	3	4	5	6	. . . .	25
a	b	c	d	e	f	g		z

A = [ ]

a	a	b	b	c	c	c	f
~~b~~	~~a~~	~~c~~	~~a~~	~~c~~	~~f~~	~~c~~	~~b~~
0	1	2	3	4	5	6	7

k

i	count	ch
0	2	a
1	2	b
2	3	c
3	0	d
4	0	e
5	1	f
⋮		

TC : O(n)

SC : O(26) ⇒ O(1)

Q. 3 Given a String, find the length of longest palindromic substring.

↳ a continous part of string

str = a b b k

total substrings ⇒

a	b	b	k
a b	b b	b k	
a b b	b b k		
a b b k			

ans : 2 (bb)

i) brute force → go on every substring and find out if that is palindromic. If it is palindromic, it can be your answer.

```
int solve ( String str) {
    int n= str.length();
    int ans = 0;

    for (int s=0; s<n ; s++) {              } → go on every substring
        for (int e=s; e<n; e++) {
            if(isPal (str,s,e) == true) {
                ans = Math.max (ans, e-s+1);
            }
        }
    }
}
```

TC: $O(n^3)$

Expected TC: $O(n^2)$

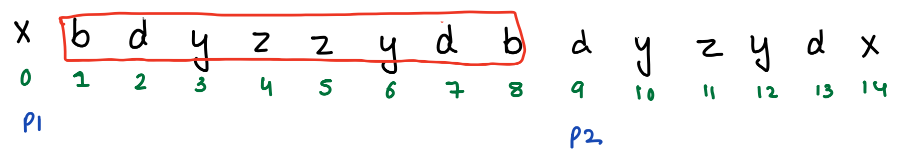
	←	←	←	←	→	→	→	→	←	←		→	→	
x	b	d	y	z	z	y	d	b	d	y	~~z~~	y	d	x
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

even length pal substrings: zz , yzzy, dyzzyd,
bdyzzydb

odd length pal substrings: z, yzy, dyzyd

TC: $O(n^2)$

```
int solve (String str) {
    int ans=1;
    int n= str.length();
    //go on 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));
    }
     //go on odd length substring
     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;
}
```

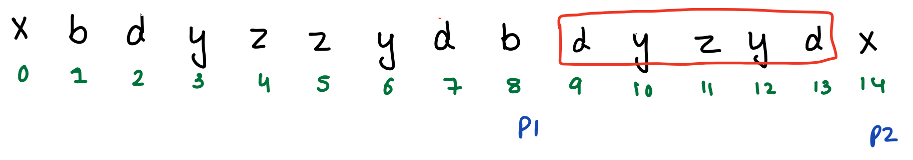


even length

$i = 4$

$p2 - p1 + 1 - 2$

$p2 - p1 - 1 \Rightarrow 9 - 0 - 1 = 8$



odd length

$i = 11$

$p2 - p1 - 1 = 14 - 8 - 1 = 5$

```
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;
}
```

dry run

```
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;
}
```

```
                                  i
str:     a   b   c   b   a   k   k   a   m
         0   1   2   3   4   5   6   7   8
                                     p1  p2
```

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

ans = ~~2~~ ~~4~~ 5

```
                                       i
str:       a   b   c   b   a   k   k   a   m
      -1   0   1   2   3   4   5   6   7   8
                                   p1      p2
```

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

How using StringBuilder over String is better in Java.

→ disadvantages of String

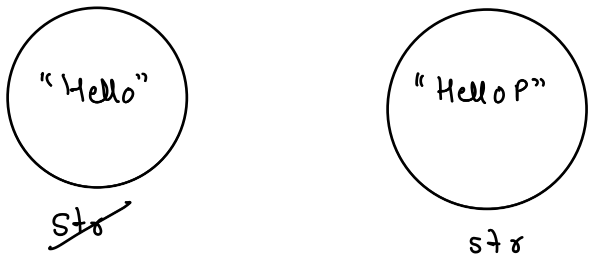
i) Strings are immutable

ii) concatenation in strings is inefficient

StringBuilder can help us to manage both of the above issues.

```
String str = "Hello";
str += 'P';
```

$\longrightarrow$ O(n)



concatenation is inefficient (in Java's string)