① $C_i = i$ if i is exact power of 2

$C_i = 1$ ~~if~~ otherwise

Let's assume we charge \$4 for each operation.

Operation index	0	1	2	3	4	5	6	7	8
Charge	4	4	4	4	4	4	4	4	4
cost	1	1	2	1	4	1	1	1	8
Interest or balance	3	6	8	11	11	14	14	20	16

Amortization cost $\sum_{i=0}^{n} 4 = 4n.$

Actual cost $\begin{cases} f(2^m) = 2^m \\ 1, \text{otherwise} \end{cases}$

thus $f(2^m) = 2m$

assume $n = 2^m$

$f(n) = n \Rightarrow n + \textcircled{c} \rightarrow$ Constant

Since c is always $< n$.

$4n > n + c$

then the Amortization cost per operation $\frac{4n}{4} = 4 \Rightarrow O(4) \Rightarrow O(n)$.

2. The java code:

```java
public class BubbleSortImproved {
    static int [] a = {2,5,1,6,9,4,42,35};
    public static void main(String [] args) {
        bubbleSort();
        for(int k=0;k<a.length;k++) {
            System.out.print(a[k]+" ");
        }
    }
    public static void bubbleSort() {
        int len = a.length;
        for(int i = 0; i < len-1; ++i) {
            if (a[i] < a[i + 1]) {
                continue;
            }
            else {
                for (int j = 0; j < len - 1; ++j) {
                    if (a[j] > a[j + 1]) {
                        swap(j, j + 1);
                    }
                }
            }
        }
    }
    static void swap(int i, int j){
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

Our code running time is O(n) because, if the array is already sorted the inner loop will never be executed.

3. The java Code

```
public class BubbleSortImproved2 {
    static int [] a = {2,5,1,6,9,4,42,35};
    public static void main(String [] args) {
        bubbleSort();
        for(int k=0;k<a.length;k++) {
            System.out.print(a[k]+" ");
        }
    }
    public static void bubbleSort() {
        int len = a.length;
        for(int i = 0; i < len-1; ++i) {
                for (int j = 0; j < len - 1-i; ++j) {
                    if (a[j] > a[j + 1]) {
                        swap(j, j + 1);
                    }
                }
            }
    }
    static void swap(int i, int j){
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

Instead of running the inner loop n times our code runs it n-i time for each iteration of the outer loop (Notice i is always increasing), Hence there will be a significant performance improvement.

(4)

```
Algorithm  Sort012 ( )
Input : Array A of n length with inputs from set {0,1,2}
Output : Array A Sorted

  Count0 ← 0;
  Count1 ← 0;
  Count2 ← 0
   n ← a.length-1
for i ← 0 to n do
    if A[i] = 0 then ~~count~~ increment Count0
    else if A[i] = 1 then increment Count1
    else . then increment Count2
For j ← 0 to count0 do
        A[j] = 0;
For j ← count0 to count0 + count1 do
        A[j] = 1;
For j ← count0 + count 1 to n do
        A[j] = 2;

return A;
```

⇒ Since we don't have nested loops (only ~~on~~ a loop that runs up to n maximum) the running time is O(n) //

4.The java code:

```java
public class Sort012 {
    static int [] a={0,1,0,2};
    public static void main(String [] args) {
        sort012();
        for(int i=0;i<a.length;i++) {
            System.out.print(a[i]+" ");
        }
    }
    public static void sort012() {
        int count0=0,count1=0,count2=0;
        for(int i=0;i<a.length;i++) {
            if(a[i]==0) count0++;
            else if(a[i]==1) count1++;
            else count2++;
        }
        for(int j=0;j<count0;j++) {
            a[j]=0;
        }
        for(int j=count0;j<(count0+count1);j++) {
            a[j]=1;
        }
        for(int j=(count0+count1);j<a.length;j++) {
            a[j]=2;
        }
    }
}
```