CS61B Week7 Note- Asymptotics

Gary Agasa

One.Operation Counting

```java
public static boolean dup1(int[] A) {
  for (int i = 0; i < A.length; i += 1) {
    for (int j = i + 1; j < A.length; j += 1) {
      if (A[i] == A[j]) {
        return true;
      }
    }
  }
  return false;
}
```

dup1

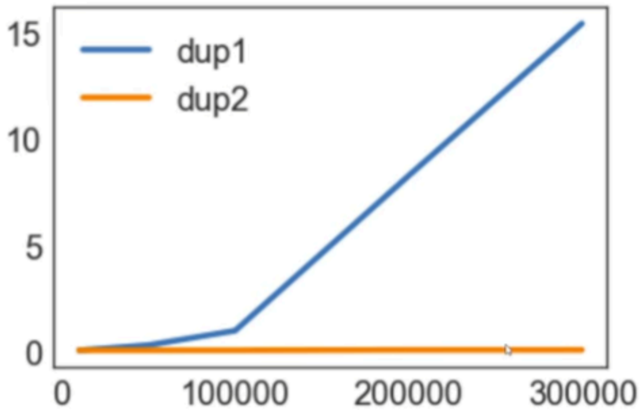
dup2

```java
public static boolean dup2(int[] A) {
  for (int i = 0; i < A.length - 1; i += 1) {
    if (A[i] == A[i + 1]) {
      return true;
    }
  }
  return false;
}
```

> 升序数组中是否有元素相同的两种算法

N	dup1	dup2
10000	0.08	0.08
50000	0.32	0.08
100000	1.00	0.08
200000	8.26	0.1
400000	15.4	0.1

Time to complete (in seconds)




```java
for (int i = 0; i < A.length; i += 1) {
  for (int j = i+1; j<A.length; j += 1) {
    if (A[i] == A[j]) {
       return true;
    }
  }
}
return false;
```

operation	symbolic count	count, N=10000
i = 0	1	1
j = i + 1	1 to N	1 to 10000
less than (<)	2 to $(N^2+3N+2)/2$	2 to 50,015,001
increment (+=1)	0 to $(N^2+N)/2$	0 to 50,005,000
equals (==)	1 to $(N^2-N)/2$	1 to 49,995,000
array accesses	2 to N^2-N	2 to 99,990,000

> Intutive Simplification：

> 1. Consider only the worst cases
> 2. Pick some representative operation to act as a proxy for the overall runtime.(The biggist increment)
> 3. Ignore the lower order terms
> 4. Ignore multitative constants

simplified growth analysis process:

1. Choose arepresentative operation to count
2. Using the intution and inspection to determine order of growth

- *e.g.*

```
int N = A.length;
for (int i = 0; i < N; i += 1)
   for (int j = i + 1; j < N; j += 1)
      if (A[i] == A[j])
         return true;
return false;
```

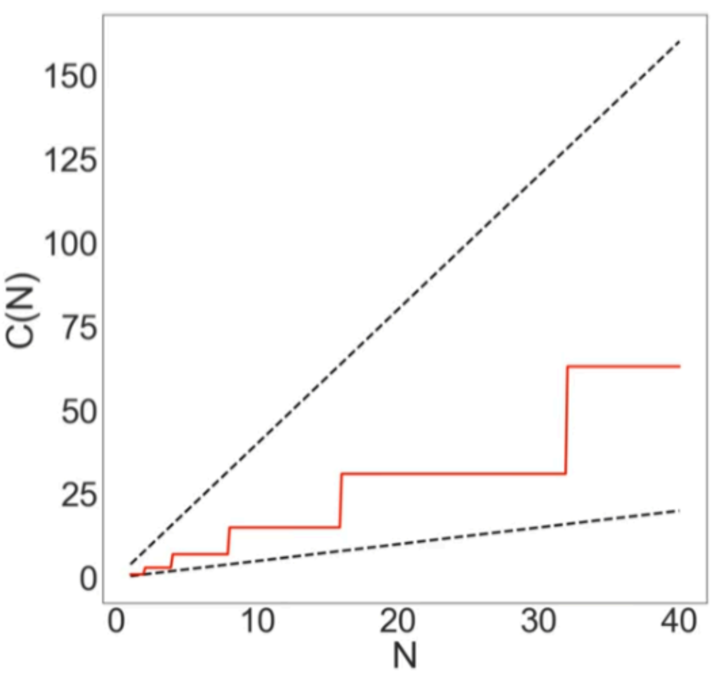
  i. choose the "==" operation as the counting model
  ii. In the worst case,the"==" operation is approximately operated $\frac{N^2}{2}$times
  iii. So $R(N) = \frac{N^2}{2} \in \Theta(N^2)$

Two.Formalizing Order of growth--Time Complexity Θ

- Defination:
  Given the program's runtime $R(N)$,$R(N) \in \Theta(f(N))$means there exists positive constants k_1 and k_2 such that:
  $k_1 f(N) \leq R(N) \leq k_2 f(N)$
  e.g
  $N^3 + 3N^4 \in \Theta(N^4)$,
  $Ne^N + N \in \Theta(Ne^N)$
  so we can conclude the time complexity of the two algorithms at the beginning of the note are $\Theta(N^2)$ and $\Theta(N)$
- A Example of Time Complexity

```
public static void printParty(int N) {
    for (int i = 1; i <= N; i = i * 2) {
        for (int j = 0; j < i; j += 1) {
            System.out.println("hello");
            int ZUG = 1 + 1;
        }
    }
}
```

> Consider when $N = 2^n$:
> take the print operation as the counting model
> times=$1 + 2 + 4 + ... + 2^{log_2(N)} = \frac{2^{log_2(N)+1} - 1}{2-1} = 2n - 1$,$R(N) \in \Theta(N)$



Reflection:Cannot simply calculate the time complexity by counting the times of nested loops

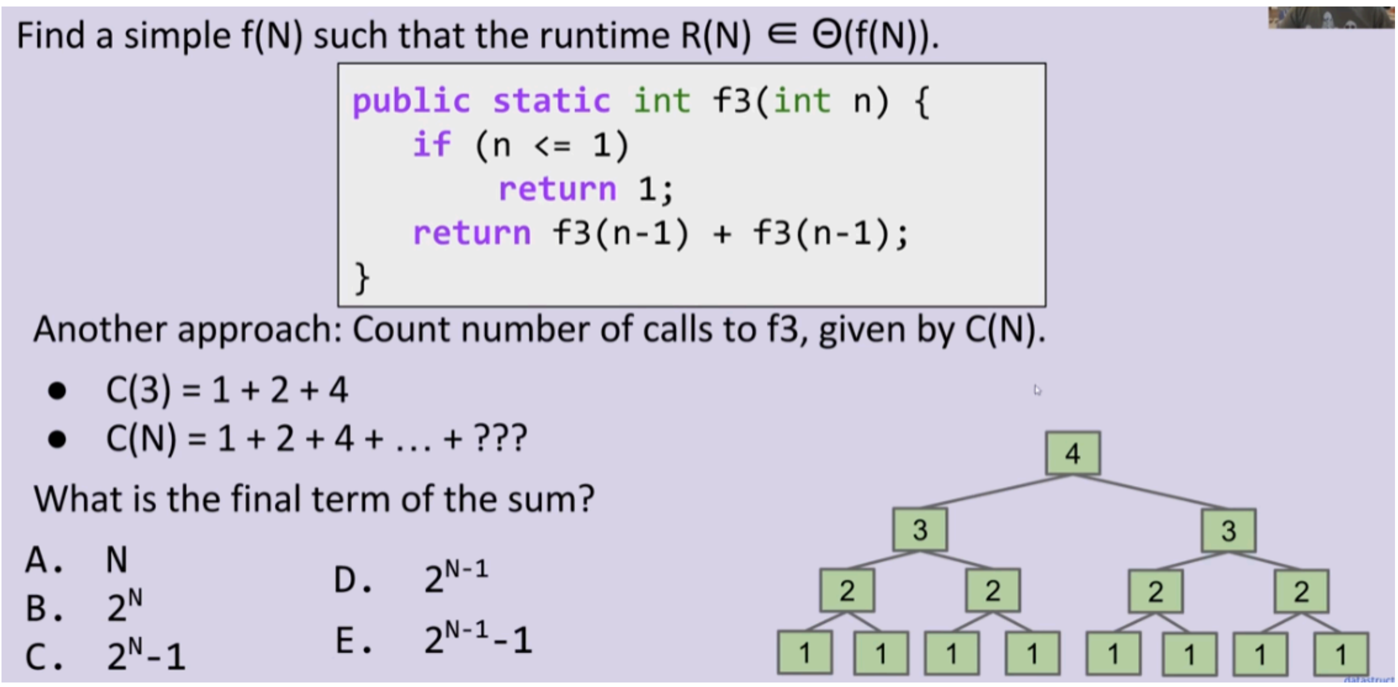
1. There are **no magic shortcut** for the problem of calculating the time complexity.
   But we can know:

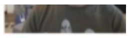
$$1 + 2 + 3 + ... + Q = \frac{Q(Q+1)}{2} \in \Theta(Q^2)$$
$$1 + 2 + 4 + 8 + ... + Q = 2Q - 1 \in \Theta(Q)$$

   to help us analize the time complexity.
   - Strategies:
     - Find the exact sum
     - Write out examples .
     - Draw pictures

Three.Some examples of Time Complexity analysis

- **Exercise1:*Tree Recursive***



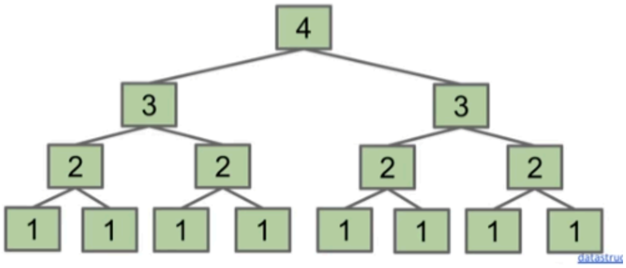


Find a simple f(N) such that the runtime R(N) $\in \Theta(f(N))$.

```java
public static int f3(int n) {
   if (n <= 1)
       return 1;
   return f3(n-1) + f3(n-1);
}
```

Another approach: Count number of calls to f3, given by C(N).

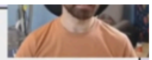
- $C(N) = 1 + 2 + 4 + \ldots + 2^{N-1}$
- Solving, we get $C(N) = 2^N - 1$



Since work during each call is constant:

- $R(N) = \Theta(2^N)$

- **Exercise2:*Binary Search***



```java
static int binarySearch(String[] sorts, String x, int lo, int hi)
    if (lo > hi) return -1;
    int m = (lo + hi) / 2;
    int cmp = x.compareTo(sorted[m]);
    if (cmp < 0) return binarySearch(sorted, x, lo, m - 1);
    else if (cmp > 0) return binarySearch(sorted, x, m + 1, hi);
    else return m;
}
```

Goal: Find runtime in terms of N = hi - lo + 1 [i.e. # of items being considered]

- Intuitively, what is the order of growth of the worst case runtime?

A. 1
B. $\log_2 N$
C. N
D. $N \log_2 N$
E. 2^N

<u>Intutive:</u>

if is the times of calls to binarysearch,solve for $1 = \frac{N}{2^C}$,so $C = log_2(N)$

<u>Exact Count:</u>

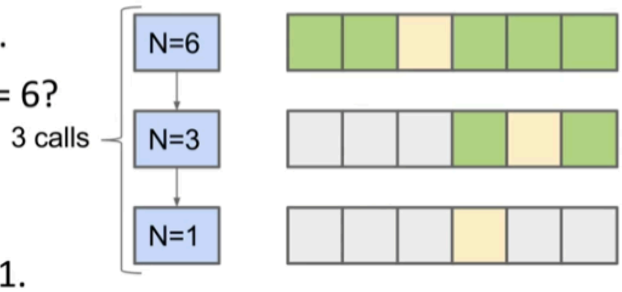
When N==6

Goal: Find worst case runtime in terms of N = hi - lo + 1 [i.e. # of items]

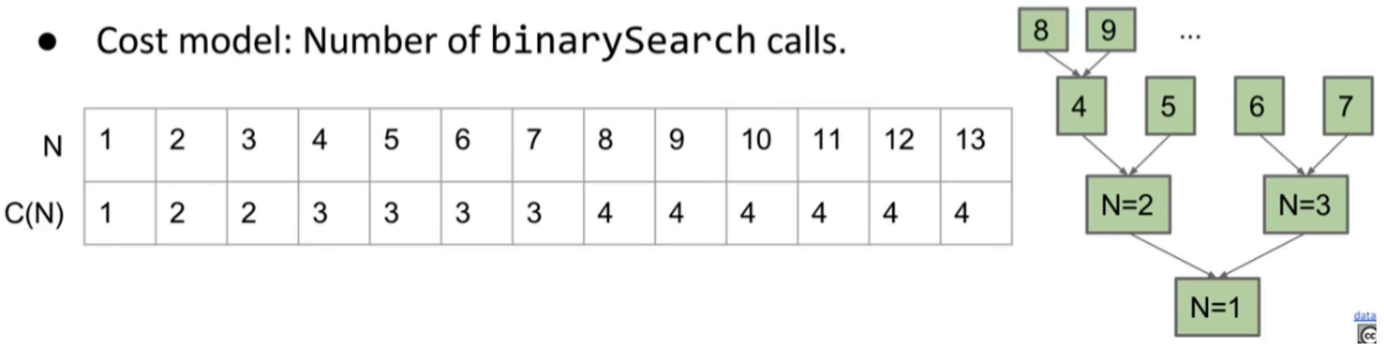
Cost model: Number of binarySearch calls.

- What is C(6), number of total calls for N = 6?

B. 3

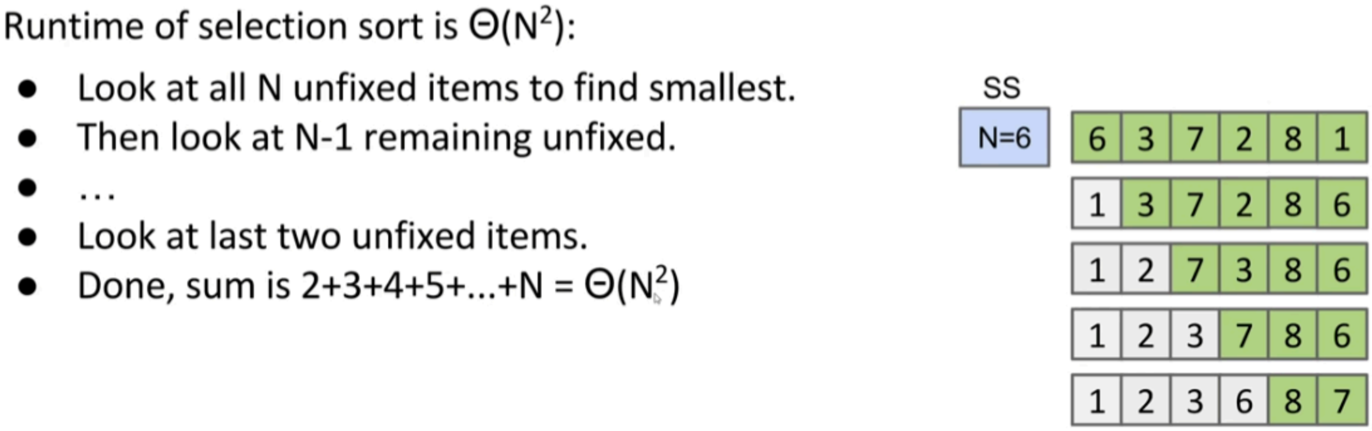


Three total calls, where N = 6, N = 3, and N = 1.

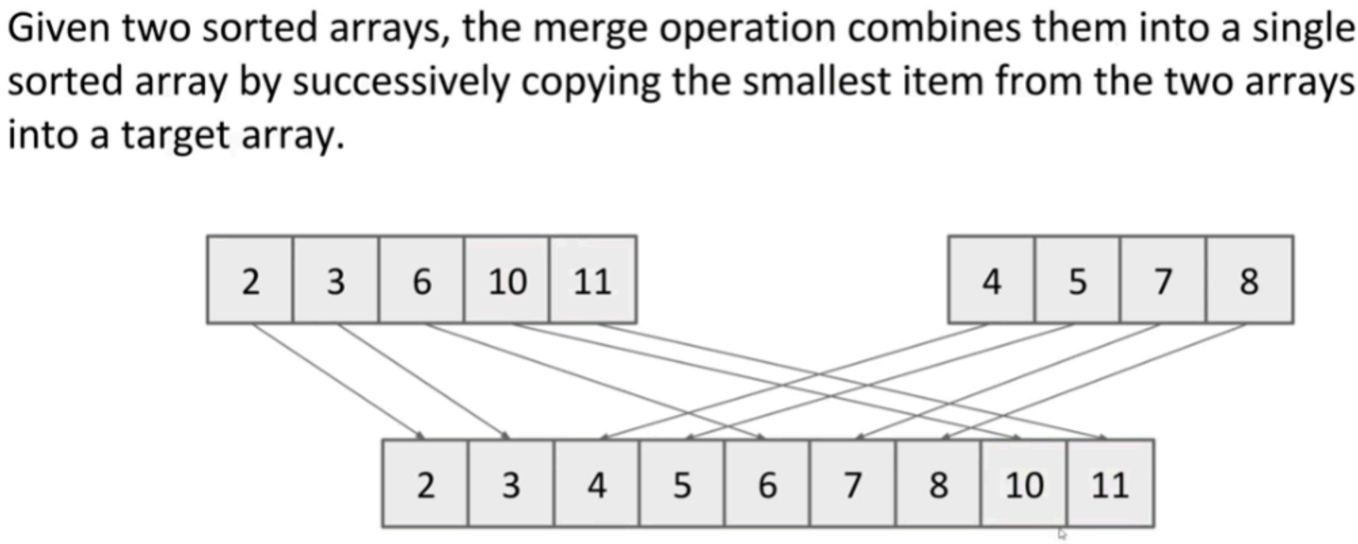


$C(N) = log_2(N) + 1 = \Theta(logN)$

- **Example3:*Selection Sort（选择排序）*** (A prelude(序曲) to Mergesort)



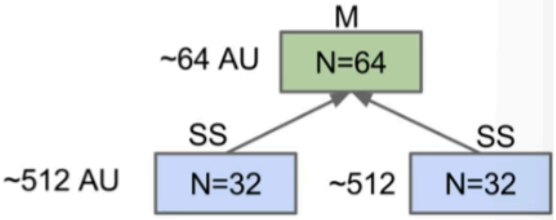
- **Example4：*Merge Sort(归并排序)***

> *Array Merging*:
>
> 
>
>
> Time Compexity:$\Theta(N)$(Use array writes as cost model)

Since the selection sort is slow ,the key idea of merge sort is to **divide the array into subarrays** to reduce the scale of selection sort to increase the speed

N=64: ~1088 AU.

- Merge: ~64 AU.
- Selection sort: ~2*512 = ~1024 AU.



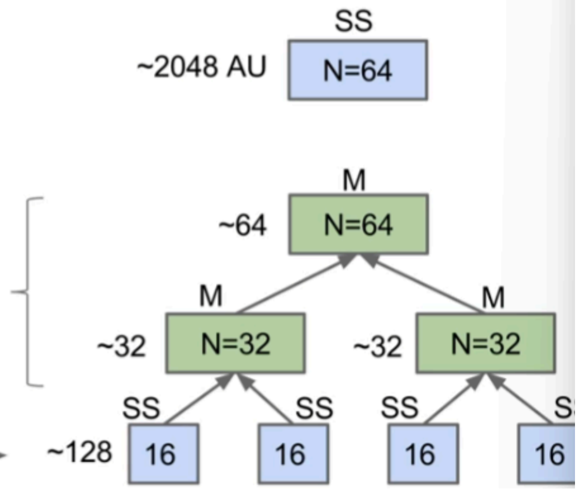
Still $\Theta(N^2)$, but faster since $N+2*(N/2)^2 < N^2$

- ~1088 vs. ~2048 AU for N=64.

Let's continue to break the array into smaller pieces:

Runtime for each sort:

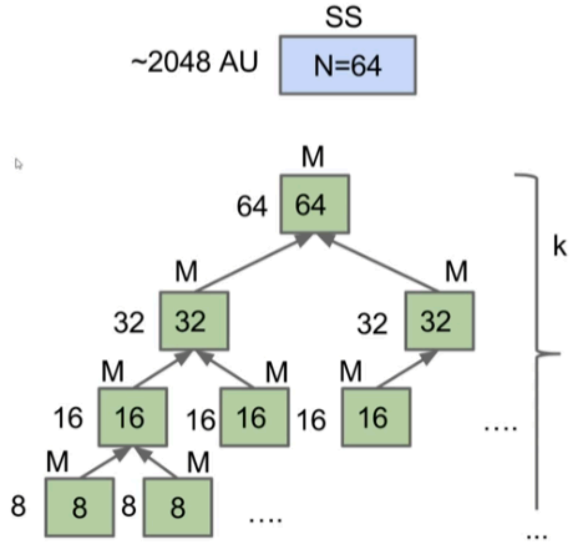
- Selection sort only: ~2048 AU.
- One layer of merges: ~1088 AU.
- Two layers of merges: ~640 AU.
  - Merge: ~64 AU + 2*~32 AU.
  - Selection sort: 4*~128.



When the subarrays' size is 1, we even don't need to sort !!!

Mergesort does merges all the way down (no selection sort):

- If array is of size 1, return.
- Mergesort the left half: $\Theta(??)$.
- Mergesort the right half: $\Theta(??)$.
- Merge the results: $\Theta(N)$.



Calculate the time compexity:

- The first layer:
  Sort:N^2,Merge:0
- The second layer:
  Sort:$2(\frac{N}{2})^2$,Merge:N

- The third layer:
  Sort:$4(\frac{N}{4})^2$,Merge:$2N$
  ...
- The k th layer,$k = log_2(N)$,where each subarray's size=1:
  Sort:$N(\frac{N}{N})^2$,Merge:$(k-1)N = (log_2(N)-1)N$
  So time complexity=$\Theta(NlogN)$

> For each layer,we conduct a merge whose scale is N. In the k th layer,we totally need to merge in a scale of (k-1)N.

Four.Big O

Whereas Θ can informally be informally be thought of as something like "equals",Big O can be thought of as "less than or equal"
e.g.

$$N^3 + 3N^4 \in O(N^4)$$
$$N^3 + 3N^4 \in O(N^6)$$
$$N^3 + 3N^4 \in O(N!)$$

1. Defination:
   $R(N) \in O(f(N))$means that there exists positive constants k_2 such that:$R(N) \le k_2 f(N)$ for all values of N greater than some N_0
2. Contrast between Θ and O
   Θ is more informative than O,but in real world we usually use O.
   e.g. We often say mergesort is $O(Nlog(N))$ rather than $\Theta(Nlog(N))$.The idea is that O

   Allows us to make simple blanket statements, e.g. can just say “binary search is O(log N)” instead of “binary search is Θ(log N) in the worst case”.

   We don't essentially need to know the exact time,but just the upper bound.

Five.Big Ω

Big Omega can be thought of as "greater than or equal"
e.g.

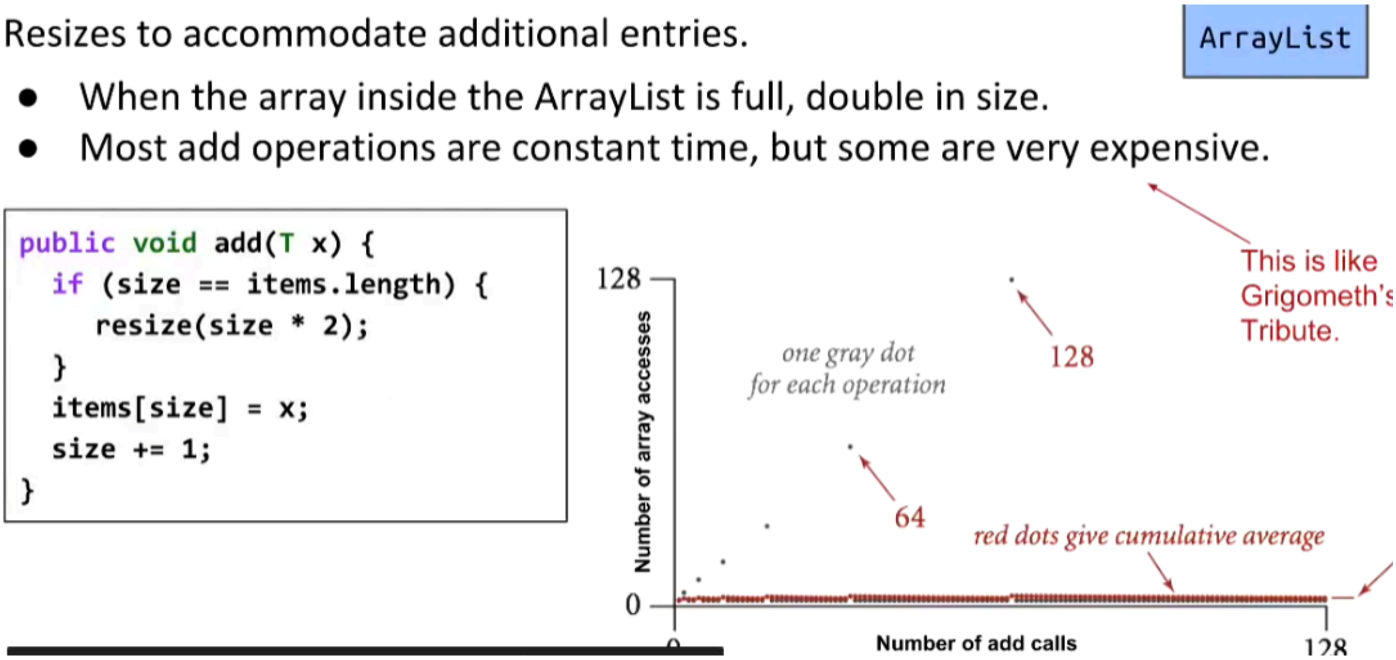
$$N^3 + 3N^4 \in \Omega(N^4)$$
$$N^3 + 3N^4 \in \Omega(N^2)$$
$$N^3 + 3N^4 \in \Omega(log(N))$$

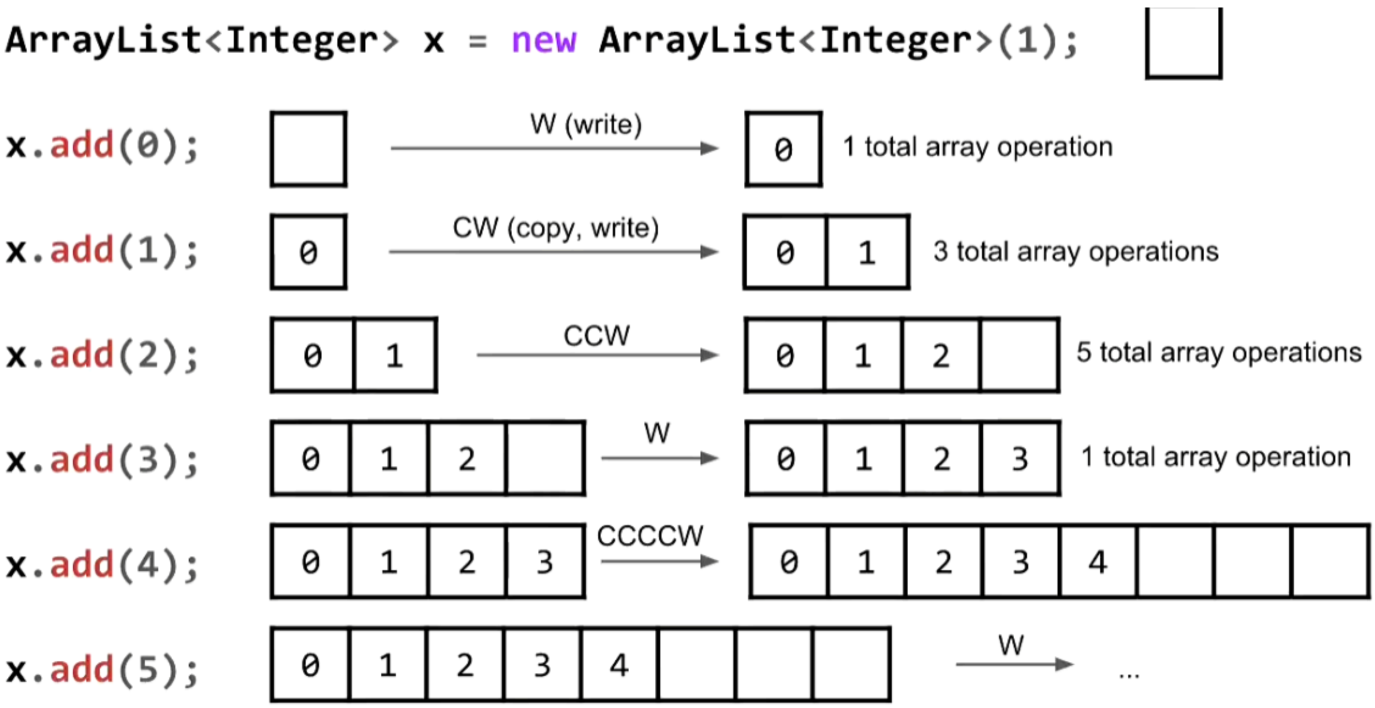
1. Defination:
   $R(N) \in \Omega(f(N))$means that there exists positive constants k_2 such that:$k_2 f(N) \le R(N)$ for all values of N greater than some N_0
2. Θ,O and Ω

	Informal meaning:	Family	Family Members
Big Theta $\Theta(f(N))$	Order of growth is f(N).	$\Theta(N^2)$	$N^2/2$ $2N^2$ $N^2 + 38N + N$
Big O $O(f(N))$	Order of growth is less than or equal to f(N).	$O(N^2)$	$N^2/2$ $2N^2$ lg(N)
Big Omega $\Omega(f(N))$	Order of growth is greater than or equal to f(N).	$\Omega(N^2)$	$N^2/2$ $2N^2$ e^N

Six.Amortized Analysis

1. Take the resizing array insert as the example:





Insert #	0	1	2	3	4	5	6	7	8	9	10	11	12	13
a[i] = cost (write cost)	1	1	1	1	1	1	1	1	1	1	1	1	1	1
resize cost(copy cost)	0	2	4	0	8	0	0	0	16	0	0	0	0	0
total cost for insert #	1	3	5	1	9	1	1	1	17	1	1	1	1	1
cumulative cost	1	4	9	10	19	20	21	22	39	40	41	42	43	44

- "Amortized(均摊的)" total cost seems to be about 44/14=3.14 acesses/item.
- R=Even though some elements cost lineral time $\Theta(N)$,average cost insert is $\Theta(1)$

2. Potentials(Φ) and Amortized Cost Bounds
   i. Defination of potential:
   Let Φ_i be the potential at time i.The potential represent the cumulative(积累的)defference between **arbitary** amortized costs and actual costs over time.
   - Let the amortized cost be 3

actual cost, ci	1	2	0	4	0	0	0	0	8	0	0	0	0	0
amortized cost, ai	3	3	3	3	3	3	3	3	3	3	3	3	3	3
change in potential	2	1	3	-1	3	3	3	3	-5	3	3	3	3	3
potential **Φ**i	2	3	6	5	8	11	14	17	12	15	18	21	24	27

- Let the amortized cost be 5

Insert #		0	1	2	3	4	5	6	7	8	9	10	11	12	13
total cost, ci		1	3	5	1	9	1	1	1	17	1	1	1	1	1
amortized cost, ai		5	5	5	5	5	5	5	5	5	5	5	5	5	5
change in potential		4	2	0	4	-4	4	4	4	-12	4	4	4	4	4
potential **Φ**i	0	4	6	6	10	6	10	14	18	6	10	14	18	22	26