2Sum and 3Sum Problems

May 16, 2019

2Sum problems

Given an array $A[1 \cdots n]$ of integers and integer m, find if there are indices i and j (not necessarily distinct!) such that $A[i] + A[j] = m$.

3Sum problems

Given an array $A[1 \cdots n]$ of integers and integer m, find if there are indices i, j and k (not necessarily distinct!) such that $A[i] + A[j] + A[k] = 0$.

Let us focus on the 2Sum Problem first. We can solve 3Sum problem by applying the algorithm of 2Sum.

First Solution for 2Sum:

Algorithm 1 Simple solution

```
1: for i = 1 to n do
2:   for j = i to n do
3:     if A[i] + A[j] = m then
4:       return true
5:     end if
6:   end for
7: end for
8: return false
```

Analyze this is $\Theta(n^2)$, it computes all the possible solution naively.

Second Solution for 2Sum:
Consider we do some work to the array.
Sorting the entire array is really helpful!

Algorithm 2 pre-sorting solution

```
1: Sort(A)
2: for i = 1 to n do
3:    j = BinarySearch(m-A[i],A)
4:    if j > 0 then
5:       return true
6:    end if
7: end for
8: return false
```

Simply time analysis:
Consider Sort(A), we can use merge sort whose complexity is $\Theta(nlog(n))$
For loop goes through the entire array that is $\Theta(n)$, one BinarySearch takes $\Theta log(n)$
Therefore the complexity of this algorithm is $\Theta(nlog(n))$

Third Solution for 2Sum:
Consider the array is pre-sorted by using merge sort. We can do better in the second part.

Algorithm 3 Fastest solution

```
1: i = 1; j = n
2: while i ≤ j do
3:    if A[i] + A[j] = m then
4:       return true
5:    else
6:       if A[i] + A[j] < m then
7:          i = i + 1
8:       else
9:          j = j + 1
10:      end if
11:   end if
12: end while
13: return false
```

Explanation:
This algorithm needs a little bit creativity.
We set i to be the index of the beginning of array and j to be the index of the end of array. Since the entire array has been pre-sorted, therefore we know that if the number we compute from $A[i]$ and $A[j]$ is less than m, we know that we need to increment i so that we can get a larger result. Vice versa (decrement j)

Solutions for 3Sum:

Solution 1:
Similar to 2Sum, we can compute the entire array by using three nested loops to check each triplet that is $\Theta(n^3)$

Solution 2:
For each value of k, use the 2Sum second solution to find i, j $A[i] + A[j] = -A[k]$
Since the k needs to go through the entire array which is $\Theta(n)$ and each time it runs the second solution for 2Sum that is $\Theta(nlogn)$, therefore the total complexity is $\Theta(n^2(logn))$.

Solution 3:
In the second solution for 3Sum, we sort the array n times, which is not efficient.
So we pre-sort the array before executing the program.

Solution 4:
The fourth algorithm is basically replace the loop by using third solution for 2Sum problem.
However this time it makes a great change!

Algorithm 4 Faster solution for 3Sum

```
1: Sort(A)
2: for i = 1 to n do
3:    x = Sorted2Sum(A, -A[k])
4:    if x then
5:       return true
6:    end if
7: end for
8: return false
```

Analysis:
Consider sort array A, we apply merge sort so that it takes $\Theta(nlogn)$
Since the loop in the third solution for 2Sum only takes $\Theta(n)$, since it goes the array only once.
Therefore we have $\Theta(n^2)$ complexity as k has to goes through the entire array as well.