HW3

1. *Solve the following recurrence equations exactly*

   a. Suppose that $n = 2^{2^k}$ for some integer $k \geq 0$.

$$\begin{aligned} &T(n) = 4\sqrt{n}T(\sqrt{n}) + n \quad n > 2 \\ &T(2) = 1 \end{aligned} \tag{1}$$

Ans:

size of subproblems	number of subproblems	Non-recursive cost
$n = 2^{2^k}$	1	$n \cdot 1 = n$
$\sqrt{n} = 2^{2^{k-1}}$	$4\sqrt{n} = 4 \cdot n/2^{2^{k-1}}$	$2^{2^k-1} \cdot 4 \cdot n/2^{2^k-1} = 4n$
$n^{1/4} = 2^{2^{k-2}}$	$4 \cdot n/2^{2^{k-1}} \cdot 4 \cdot 2^{2^{k-1}}/2^{2^{k-2}} = 4^2 \cdot n/2^{2^{k-2}}$	$2^{2^k-2} \cdot 4^2 \cdot n/2^{2^k-2} = 4^2 n$
	...	
$4 = 2^{2^1}$	$4^{k-1} \cdot n/2^{2^1}$	$2^{2^1} \cdot 4^{k-1} \cdot n/2^{2^1} = 4^{k-1}n$
$2 = 2^{2^0}$	$4^k \cdot n/2^{2^0}$	$1 \cdot 4^k \cdot n/2^{2^0} = 4^k n/2$

> The total cost is $n(1 + 4 + 4^2 + \cdots + 4^{k-1}) + 4^k n/2 = n \cdot \frac{1 \cdot (1-4^k)}{1-4} + 4^k n/2 = \frac{5}{6} \cdot n \log^2 n - \frac{1}{3}n.$
>
> Check. Base case: $n = 2, k = 0$. Our solution gives $T(1) = \frac{5}{3} - \frac{2}{3} = 1$, which is correct.
> The next larger value of n: $n = 4, k = 1$. Our solution gives $T(4) = \frac{40}{3} - \frac{4}{3} = 12$, which is also correct.

   b. Suppose that $n = 2^k$ for some integer $k \geq 0$.

$$\begin{aligned} &T(n) = T(n/2) + \log n \quad n > 1 \\ &T(1) = 1 \end{aligned} \tag{9}$$

Ans:

size of subproblems	number of subproblems	Non-recursive cost
$n = 2^k$	1	$\log n$
$n/2 = 2^{k-1}$	1	$\log n/2$
$n/2^2 = 2^{k-2}$	1	$\log n/2^2$
	...	
$n/2^{k-1} = 2^1$	1	$\log n/2^{k-1}$
$n/2^k = 2^0$	1	1

> The total cost is
>
> $$\begin{aligned} &\log n + \log n/2 + \log n/2^2 + \cdots + \log n/2^{k-1} + 1 \\ &= k \cdot \log n - (\log 2 + \log 2^2 + \cdots + \log 2^{k-1}) + 1 \\ &= k \cdot \log n - (1 + 2 + \cdots + (k-1)) + 1 \\ &= k \cdot \log n - (k-1)(1 + k - 1)/2 + 1 \\ &= 3/2 \log^2 n + 1/2 \log n + 1 \end{aligned} \tag{10}$$
>
> Check. Base case: $n = 1, k = 0$. Our solution gives $T(1) = 0 + 0 + 1 = 1$, which is correct.
> The next larger value of n: $n = 2, k = 1$. Our solution gives $T(2) = \frac{3}{2} + \frac{1}{2} = 2$, which is also correct.

2. *Let $A[1:n]$ be an array of integer values, which we view as points on a line. The task is to determine how many pairs of points are distance d or less apart; d is an input parameter. Give an $O(n \log n)$ time algorithm to solve this problem. Remember to analyze the running time of your algorithm.*

> Define a function $POD(d, A[1:n])$ to count the number of pairs of points are distance d or less apart
>
> Just to clarify: n is the length of the array; the index starts from 0 to n-1; currentIn is to indicate if all the points in the array between left and right is less or equal to d.

> *The idea of the algorithm is continually increasing left and right variable to iterate all of the points in the array to find the total number:*
>
> 1. *If $A[right] - A[left] <= d$, it might be following situations:*
>   1. *The left and right may be same, so this is not a pair we need*
>   2. *As right increases, an extra pair is found.*
> 2. *if $A[right] - A[left] > d$, it might be following situations, we have to add this number of pairs to num:*
>   1. *The array between left to right-1 are pairs whose distance is less or equal to d*
>   2. *The array between left to right-1 are pairs whose distance is still larger than d*
>
> *So everytime $A[right] - A[left] > d$, we check if we need to plus a number to num.*
>
> *As while loop will break when right is larger than n, there may be a situation that the currentIn is true, which means array between left and right are all less or equal to d. So it can be simply added by summation formula of arithmetic sequence.*
>
> The Sort() function is to sort the values in the array in an increasing order, whose time complexity if $O(n \log n)$
>
> The line 2 and line 3 will have at least 5 operations
>
> The while loop: left starts from 0 to n, right starts from 0 to n. As we can see, every line in the if condition is c operations and every step in while loop, either left or right will plus one, so at most execute 2n times, the while loop will end.
>
> Hence the time complexity will be $O(n \log n) + O(n) = O(n \log n)$

```
POD(d,A[1:n]):
  Sort(A[1:n])
  if (n < 2) then return 0
  left<-0, right<-1, num<-0, currentIn<-false
  while(right<n) do
    if(A[right]-A[left]<=d) do
      if(right!=left) then currentIn<-true
      right<-right+1
    else if(A[right]-A[left]>d) do
      if(currentIn) do
        num<-num+right-left
        currentIn<-false
      left<-left+1;
    end if
  if (currentIn) then num <- num+ (right-left)(1+right-left)/2
  end while
```

3. *Suppose you are given an array A[d : u] of distinct integers, and an integer r in the range [1, u−d+ 1]. Suppose further that you have an algorithm ApproxSelect(A, d, u, r, i, k). Then, using ApproxSelect as a subroutine, give an algorithm Find(A, d, u, r) to find the r-th smallest item in an array A[d : u] running in worst case time O(n). You should describe your algorithm and analyze its running time.*

> The r is always between i and k. When $i = k$, r would equal to the i or k.
>
> Therefore, first, run $ApproxSelect()$, if $i! = k$, go into the loop, otherwise, return $A[i]$, which is the results r.
>
> In the while loop, we keep running $ApproxSelect()$ and modify the boundary of array A, until i equals to k
>
> Recurrence equations would be (constant c is ignored, use 1 to replace):
>
> $$\begin{aligned} &T(n) = T(n/2) + 1 \quad n > 1 \\ &T(1) = 1 \end{aligned} \tag{11}$$
>
> Hence, the time complexity would be $O(n)$. First, we run $ApproxSelect()$, which runs in worst-case time $O(n)$. When in while loop, each step we have c operation and execute $\log n$ times, and $ApproxSelect()$ runs in $n/2$ cases. Hence, the time complexity would be $O(n) + O(c \log n) + O(n) = O(n)$

```
Find(A,d,u,r)
  ApproxSelect(A,d,u,r,i,k)
  while(i<k)
    d<-i,u<-k
    ApproxSelect(A,d,u,r,i,k)
  end while
  return A[i]
```

4. *Let $0 \le i < n/2$. Show that $b_i = \left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^i\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i$, and $b_{n/2+i} = \left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^{n/2+i}\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i$. The index i refers to the i th entry in a vector.*

   Analyze the runtime of your algorithm. You may assume that all arithmetic operations take constant time.

   $V_{n/2}(x^2)$ is shown below:

$$V_{n/2}(x^2) = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & x^2 & x^4 & \dots & x^{n/2-1} \\ 1 & x^4 & x^8 & \dots & x^{2(n/2-1)} \\ & & \dots & & \\ 1 & x^{n/2-1} & x^{2(n/2-1)} & \dots & x^{2(n/2-1)^2} \end{bmatrix} \tag{12}$$

   Show that $b_i = \left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^i\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i$:

   > $x^{2ij}a_{2j}$ stands for the even entry of the vector b_i, $x^{2ij+1}a_{2j+1}$ stands for the even entry of the vector b_i.

$$\begin{aligned}
&\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^i\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i \quad 0 \le i < n/2 \\
&= \left[\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & x^2 & x^4 & \dots & x^{n/2-1} \\ 1 & x^4 & x^8 & \dots & x^{2(n/2-1)} \\ & & \dots & & \\ 1 & x^{n/2-1} & x^{2(n/2-1)} & \dots & x^{2(n/2-1)^2} \end{bmatrix}\begin{bmatrix} a_0 \\ a_2 \\ \dots \\ a_{n-2} \end{bmatrix} + x^i \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & x^2 & x^4 & \dots & x^{n/2-1} \\ 1 & x^4 & x^8 & \dots & x^{2(n/2-1)} \\ & & \dots & & \\ 1 & x^{n/2-1} & x^{2(n/2-1)} & \dots & x^{2(n/2-1)^2} \end{bmatrix}\begin{bmatrix} a_1 \\ a_3 \\ \dots \\ a_{n-1} \end{bmatrix}\right]_i \\
&= \sum_{j=0}^{n/2-1} x^{2ij}a_{2j} + x^i \sum_{j=0}^{n/2-1} x^{2ij}a_{2j+1} \\
&= \sum_{j=0}^{n/2-1} x^{2ij}a_{2j} + \sum_{j=0}^{n/2-1} x^{2ij+i}a_{2j+1} \\
&= \sum_{j=0}^{n/2-1} \left(x^{2ij}a_{2j} + x^{2ij+i}a_{2j+1}\right) \\
&= \sum_{j=0}^{n} x^{ij}a_j = b_i
\end{aligned} \tag{13}$$

   Show that $b_{n/2+i} = \left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^{n/2+i}\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i$.

   > The $n/2+i$ row of the $V_n(x)$ is

$$\begin{bmatrix} 1 & x^{n/2} & x^n & \dots & x^{n/2(n-1)} \\ 1 & x^{n/2+1} & x^{n+2} & \dots & x^{(n/2+1)(n-1)} \\ & & \dots & & \\ 1 & x^{n-1} & x^{2(n-1)} & \dots & x^{(n-1)^2} \end{bmatrix} \tag{14}$$

$$
\begin{aligned}
&\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^{n/2+i}\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i \quad 0 \le i < n/2 \\
&= \left[\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & x^2 & x^4 & \dots & x^{n/2-1} \\ 1 & x^4 & x^8 & \dots & x^{2(n/2-1)} \\ & & \dots & & \\ 1 & x^{n/2-1} & x^{2(n/2-1)} & \dots & x^{2(n/2-1)^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_2 \\ \dots \\ a_{n-2} \end{bmatrix} + x^{n/2+i} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & x^2 & x^4 & \dots & x^{n/2-1} \\ 1 & x^4 & x^8 & \dots & x^{2(n/2-1)} \\ & & \dots & & \\ 1 & x^{n/2-1} & x^{2(n/2-1)} & \dots & x^{2(n/2-1)^2} \end{bmatrix} \begin{bmatrix} a_1 \\ a_3 \\ \dots \\ a_{n-1} \end{bmatrix}\right]_i \\
&= \sum_{j=0}^{n/2-1} x^{2ij} a_{2j} + x^{n/2+i} \sum_{j=0}^{n/2-1} x^{2ij} a_{2j+1} \\
&= \sum_{j=0}^{n/2-1} x^{2ij} a_{2j} + \sum_{j=0}^{n/2-1} x^{2ij+n/2+i} a_{2j+1} \\
&= \sum_{j=0}^{n/2-1} \left(x^{2ij} a_{2j} + x^{2ij+n/2+i} a_{2j+1}\right)
\end{aligned}
$$

$$
\begin{aligned}
b_{n/2+i} &= \sum_{j=0}^{n} x^{(n/2+i)j} a_j \\
&= \sum_{j=0}^{n/2-1} \left(x^{2(i+n/2)j} a_{2j} + x^{(i+n/2)(2j+1)} a_{2j+1}\right) \\
&= \sum_{j=0}^{n/2-1} \left(x^{2(i+n/2)j} a_{2j} + x^{2ij+i+jn+n/2} a_{2j+1}\right) \\
&= \sum_{j=0}^{n/2-1} \left(x^{2ij} x^{nj} a_{2j} + x^{2ij+i+n/2} x^{nj} a_{2j+1}\right) \\
&\text{As } x^n = 1, \\
&= \sum_{j=0}^{n/2-1} \left(x^{2ij} a_{2j} + x^{2ij+i+n/2} a_{2j+1}\right) \\
&= \left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^{n/2+i}\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i
\end{aligned}
$$

(

> For matrix calculations, the time complexity is $O(n^2)$.
>
> While we can divide the n times of calculations of b_i into (1) $n/2$ times of b_i from 0 to $n/2$ and (2) $n/2$ times of b_i from $n/2$ to n.
>
> In this case, we divide the original calculation, let's say $MultiMv(V(x), n, power, r)$, time complexity cn^2 into $c((n/2)^2)$, and continue dividing it.
>
> For each b_i, the cost $\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^e\right]_i + x^i\left[V_{n/2}\left(x^2\right)\boldsymbol{a}^o\right]_i$ is $cn/2$ depends on the V_t, where k is the label of the V.
>
> So add the all divided parts, it would be $c(1+2+4+\cdots+n/2+n) = T(n\log n)$, where the summation is k times as $n = 2^k$.
>
> The pseudo-code shows the idea:
>
> Every recursion divides the n into 2 parts: (1) the first starts with $i = 0$, the power of the parameter of V multiply 2, for example, the origin V is $V(x)$, then becomes $V(x^2)$, the result of the $MultiMV()$ return to r. (2) The second part starts with $i = n/2 + i$, and others are the same.
>
> The merge method concatenates two vectors.

```
MultiMV(V, n, i, power=1, r)
  MultiMV(V, n/2, 0, 2*power, r1)
  MultiMV(V, n/2, n/2+i, 2*power, r2)
  Merge(r1,r2)
```