CS170–Fall 2014 — Solutions to Homework 3

Quoc Thai Nguyen Truong, SID 24547327, `cs170-ig`

April 17, 2016

Collaborators: Shiv Sundram, Hriday Kemburu, Michael Ross.

1. Practice with recurrence relations

(a) $\Theta(n)$

(b) $\Theta(\log(n))$

(c) $\Theta(\log(n))$

(d) $\Theta(n)$

(e) $\Theta(n^{0.5})$

(f) $\Theta(n)$

(g) $\Theta(n^{2.41})$

(h) $\Theta(2^n)$

2. Procedural Terrain Generation

(a) Let T(n) is the asymptotic runtime of function MidpointDisplacement, as a function of n.
Since MidpointDisplacement call FillIn, we can see that it takes $\Theta(1)$ to assign the values for $H[x][y]$, $H[l][y]$, $H[x][t]$, $H[r][y]$, $H[x][b]$. There are 4 recursive calls on line 9, 10, 11, and 12, which split the problem with size of $\frac{n}{2}$. Therefore, the asymptotic runtime of Midpoint Displacement(n) is:

$$T(n) = 4T\left(\frac{n}{2}\right) + 1$$

from master theorem:

$$\Longrightarrow \boxed{T(n) = \Theta(n^2)}$$

(b) There is $\boxed{No}$ algorithm which computes the sames values for H as midpointDisplacement(). In order to fill in all the grids which is total of n^2 points, it need to fill out all n^2 points which take $\Theta(n^2)$. Therefore, there is no other algorithm can do faster.

3. Dominated?

Explanation: Let A be the collection of n points. At first, we will sorted A by increasing order value of x.
For the base case, if n = 1 the algorithm, return False. If n = 2, return True if the values of x and y of the 1st point is less than or equals to the values of x and y of the 2nd point, otherwise return False.

If $n \geqslant 3$, split the array into halves of n (boolean: left, right), recursively on left and right with size of $\frac{n}{2}$ for each, left and right will be value True if there is a solution or False otherwise.
Also,we define a boolean value "cross" which will be true or false by find the point with the lowest value of y from point 0 to point $\lfloor(\frac{n}{2})-1\rfloor$ and compare with points from point $\lfloor(\frac{n}{2})\rfloor$ to $n-1$
Finally, the algorithm will return true if any value of left, right, or cross is true. Otherwise, return false if all of them are false.

Algorithm: Let A is collection of n points
$A = [(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})]$
Sorted(A) by value of x (increasing order)

```
Line 0: Function($A$):
Line 1:     If n = 1: return False
Line 2:     If n = 2:
Line 3:         If $x_0 \leqslant x_1$ and $y_1 \leqslant y_1$: return True
Line 4:         Else: return False
Line 5:     Else:
Line 6:         left = $Function([(x_0, y_0), (x_1, y_1), \cdots, (x_{\lfloor(\frac{n}{2})-1\rfloor}, y_{\lfloor(\frac{n}{2})-1\rfloor})])$
Line 7:         right = $Function([(x_{\lfloor(\frac{n}{2})\rfloor}, y_{\lfloor(\frac{n}{2})\rfloor}), (x_{\lfloor(\frac{n}{2})+1\rfloor}, y_{\lfloor(\frac{n}{2})+1\rfloor}), \cdots, (x_{n-1}, y_{n-1})])$
Line 8:         yMinPoint = find a point with the lowest value of $y_i$,where i go from 0 to $\lfloor(\frac{n}{2})-1\rfloor$
Line 9:         cross = is true only if x and y value of yMinPoint is $\leqslant$ $x_i$ *and* $y_i$, where i go from $\lfloor(\frac{n}{2})\rfloor$ to $n-1$
Line 10:        Return left or right or cross
```

Running time:
Let T(n) be the run time of the algorithm with size of n points.

Sorting the point by the value of x in increasing order will take $\Theta(n\log(n)$. There are 2 recursive calls at line 6 and 7 on size of $\frac{1}{2}$ of points (the input of the function). Also, finding the boolean value "cross" will take $\Theta(n)$ because finding the point with lowest y value for the 1 st half will take $\frac{n}{2}$ (n is the points of input function), and checking with 2nd will take $\frac{n}{2}$.
Therefore,

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

from Master theorem:

$$\Rightarrow T(n) = \Theta(n\log(n))$$

Since we sorted at the 1st time, the Total running would be

$$\Theta(n\log(n)) + \Theta(n\log(n))$$

$$\boxed{\Theta(n\log(n))}$$

Proof of correctness:
_ Invariant:

At the end of each loop on k recursive call ($1 \leqslant k \leqslant log(n)$), the function return true if only if there exists a pair of points $(x_i, y_i), (x_j, y_j)$ such that $x_i \leqslant x_j$ and $y_i \leqslant y_j$, otherwise it return false. And the values (True, False) will be in $left$, $right$, $cross$

_ Base case: Assume there is exist a pair of points $(x_i, y_i), (x_j, y_j)$ such that $x_i \leqslant x_j$ and $y_i \leqslant y_j$. The function takes in the sorted (increasing order) x value. It splits the list into $\frac{n}{2}$, one half on left side and other half on the right side. We also find there also exist a pair of solution by finding a point with the lowest value of y on the left side , and compare that point with every point on the right side, and store the value in *cross*. We can see that the if the solution pair is exist, it will be either on the $left$ side, *right* side, or *cross*. So the Invariant is true for the base case.

_ Inductive Step: Assume that the Invariant is true at k level, we prove that it's also true for $k+1$ level.
At k level, we know that the solution pair of points must be in either *right* side ($\frac{n}{2^k}$ *points*), $left$ side ($\frac{n}{2^k}$ *points*), or *cross*. Therefore, at $k+1$ level, it must be in either in *right* side ($\frac{n}{2^{k+1}}$ *points*), $left$ side ($\frac{n}{2^{k+1}}$ *points*), or *cross*. Hence, the value true must be the return value in either $left$, *right*, *or cross*.

Therefore, it's true for $k+1$, so it's true for all k ($1 \leqslant k \leqslant \log(n)$). $\Rightarrow$ The invariant is inductive.

_ Prove correctness property:

At the $\log(n)$ level where the algorithm hits the base case $n = 1$ or $n = 2$, all the recursive calls will pop-up to level 0, return the value either True if there is a solution pair of point $(x_i, y_i), (x_j, y_j)$ such that $x_i \leqslant x_j$ and $y_i \leqslant y_j$, otherwise, it return False. We know that the values (True or False) must be the return value $left$, $right$, $andcross$, and it return True (exist a pair solution of points) if any of these ($left$, $right$, *and cross*) is true, otherwise false if all of them are False (no pair of points solution). Therefore, the algorithm is true for determine if there is exists a pair of point $(x_i, y_i), (x_j, y_j)$ such that $x_i \leqslant x_j$ and $y_i \leqslant y_j$

4. Chip design

(a) If n = 1, $H(1) = 0$
If n = 2, $H(2) = H(1) + 1 = 1$
If n = 4, $H(4) = H(2) + 1 = 2$
If n = 8, $H(8) = H(4) + 1 = 3$
Building n leaves base from $\frac{n}{2}$ leaves will create a new node which connect with two $\frac{n}{2}$, one on left and one of right which add 1 more height. Therefore:

$$H(n) = H\left(\frac{n}{2}\right) + 1$$

from the master theorem:

$$\boxed{H(n) = log(n)}$$

(b) Building n leaves base from $\frac{n}{2}$ leaves will create a new node which connect with two $\frac{n}{2}$, one on left and one of right which adding 2 more squares for the width. Hence, the width of n will be the width of two width of $\frac{n}{2}$ plus 2 squares. Therefore:

$$W(n) = 2W\left(\frac{n}{2}\right) + 2$$

$$\boxed{W(n) = 2W\left(\frac{n}{2}\right) + \Theta(1)}$$

(c)

$$Since\ W(n) = 2W\left(\frac{n}{2}\right) + \Theta(1)$$

from the master theorem

$$\boxed{W(n) = \Theta(n)}$$

$$\boxed{f(n) = n}$$

(d)

$$A(n) = H(n) \times W(n)$$

$$A(n) = \Theta(\log(n)) \times \Theta(n)$$

$$\boxed{A(n) = \Theta(n\log(n))}$$

$$\boxed{f(n) = n\log(n)}$$

(e) If n = 1, $L(1) = 1$
If n = 4, $L(4) = 2L(1) + 2 = 4$
If n = 16, $L(16) = 2L(4) + 2 = 10$ Building n leaves from $\frac{n}{4}$ leaves will create an "H" shape with 3 new nodes, 2 nodes on the intersections of "H" shape and 1 node on the intersection of middle of "H" shape, and top left , top right, bottom left and bottom right of the "H" shape will connect to "H" of $\frac{n}{2}$ leaves.Hence, building n leaves from $\frac{n}{4}$ leaves will create 2 more squares. Therefore:

$$L(n) = 2L\left(\frac{n}{4}\right) + 2$$

$$\boxed{L(n) = 2L\left(\frac{n}{4}\right) + \Theta(1)}$$

(f)

$$Since\ L(n) = 2L\left(\frac{n}{4}\right) + \Theta(1)$$

From the master theorem:

$$\boxed{L(n) = \Theta(n^{0.5})}$$

$$\boxed{f(n) = n^{0.5}}$$

(g)

$$A'(n) = L(n)^2$$

$$A'(n) = \Theta(n^{0.5})^2$$

$$\boxed{A'(n) = \Theta(n)}$$

$$\boxed{f(n) = n}$$

(h) Since Charlene's layour depicted in part(a) take $\Theta(n\log(n))$ and my layout from part(e) take $\Theta(n)$, so $\boxed{my\ layout\ from\ part(e)\ is\ better}$

5. Pattern matching, with tolerance for noise

(a)
```
Line 0: Matching(string s, string t, k):
Line 1:     m = bits long of s ; n = bits long of t
Line 2:     indices = []
Line 3:     For $i := 0$ to $n-1$:
Line 4:         errors = 0
Line 5:         For $j := 0$ to $m-1$:
Line 6:             If $t[i+j] = s[j]$ : $errors = errors + 1$
Line 7:         If $errors \leqslant k$ :
Line 8:             indices.add(i)
Line 9:     If $len(indices) > 0$: print "Yes" and return indices
Line 10:     Else: print "No"
```

(b) Before choose polynomials p(x) and q(x), we should understand how to get the coefficient of x^{m-1+i} in $p(x)q(x)$. Let say we have an coefficient C_5 which is C_5x^5 of $p(x)q(x)$. We can represent C_5 by summing all the coefficient that have the same exponential value of 5.

$$C_5x^5 = x^0x^5 + x^1x^4 + \cdots + x^5x^0$$

which is equivalent to:

$$C_5x^5 = \sum_{i=0}^{5} p_iq_{5-i}, \ where\ x^5 = x^ix^{5-i}$$

Therefore,

$$C_{m-1+i} = \sum_{j=0}^{m-1} p_jq_{m-1+i-j}, \ where\ i\ the\ index\ of\ bits\ string\ t$$

Hence, the multiplication will be $s_0\ and\ t_{i+(m-1)}$, $s_1\ and\ t_{i-1+(m-1)}\ \cdots$ $s_{m-1}\ and\ t_0$

Since we want the multiplication to be the multiplication will be $s_0\ and\ t_i$, $s_1\ and\ t_{i+1}\ \cdots\ s_{m-1}\ and\ t_{i+(m-1)}$, we have to reserve either string s or t. Thanks for the hint, we will change all the bits 0 to value -1 and keep all bits 1 to value 1. In that way, we can define how many differ bits we have between string s and string t by getting the value of coefficient ,and set it equals to $m - 2d(i)$,then solve for d to find the differ, and find the index of t from the exponential value of x correspond to that coefficient.

Therefore, Here how to choose polynomials p(x) and q(x).
change all bits 0 to -1 in string t and construct a polynomials q(x) degree of $n - 1$ from it. Example: t = 1 1 0 1 would be $q(x) = 1 + x - x^2 + x^3$

Reverse bit string s and change all bits 0 to -1 and construct a polynomials p(x) degree $m - 1$ from it. Example: s = 1 1 0 , $s\prime =$ reverse s = 0 1 1 , and $p(x) = -1 + x + x^2$

Using FFT to do p(x)q(x), and then look through all the coefficient of p(x)q(x) and exponential value of x correspond to the coefficient. We want to make sure that these coefficient in form of $m - 2d$ and $d \leqslant k$, and these exponential value in form of $m - 1 + i$ and i is the valid index.

(c) **Explanation:**
From part (b), we know how to choose polynomials for p(x) and q(x). Now, let describe an $\Theta(n \log(n))$ algorithm for this string matching problem.

We know that s is m bits long , and t is n bits long, $m < n$.
We change all bits 0 to -1 value of string t and construct polynomial q(x) degree of $n - 1$ from it.
Same for string s, plus that we reverse string s, then change all bits 0 to -1 value of string s, and construct polynomial p(x) degree of $m - 1$ from it.
Using FFT to find the product of p(x)q(x) which take $\Theta(n \log(n))$ since $m < n$. and then look through all the coefficient of p(x)q(x) and exponential value of x correspond to the coefficient. We want to make sure that these coefficient in form of $m - 2d$ and $d \leqslant k$, and these exponential value in form of $m - 1 + i$ and i is the valid index.

Here is how to check for valid coefficient and index:
Let $e =$ exponential value, where x^e and $e = m - 1 + i$, i is the index of t.

We know that $0 \leqslant i \leqslant n - m$ since the last invalid i must go from i to $i + m$. Hence,

$$0 \leqslant i \leqslant n - m$$

$$0 + m - 1 \leqslant i + m - 1 \leqslant n - m + m - 1$$

$$m - 1 \leqslant m - 1 + i \leqslant n - 1$$

$$\boxed{m - 1 \leqslant e \leqslant n - 1} \; (1)$$

Let $c =$ coefficient value of $x^{(m-1+i)}$ in p(x)q(x), we know that

$$c = m - 2d(i), \; where\ d(i)\ is\ number\ of\ bits\ differ$$

$$\boxed{d(i) = \frac{m - c}{2}} \; (2)$$

If (1) and (2) are true, and we find the index of t:

$$e = m - 1 + i$$

$$\boxed{i = e - (m - 1)}$$

then add i to the list of indices that we want to return.
Finally, check for the list of indices, if list is not empty, we know that s occurs as a substring of t with $\leqslant k$ errors at some indices, so we just output "Yes" and return the list of indices. Otherwise, the list if empty, and we output "No".

Algorithm:

```
Line 0: Matching(string s, string t, k)
Line 1:     m = number bits of s, n = number bits of t , indices = empty array
Line 2:     $t\prime$ = change all bits 0 to -1 of string t
Line 3:     $s\prime$ = reverse s and change all bits 0 to -1
Line 4:     $p(x)$ = construct polynomial degree of $m - 1$ from $s\prime$ # 1 1 -1 = $1 + x - x^2$
Line 5:     $q(x)$ = construct polynomial degree of $n - 1$ from $t\prime$ # -1 1 -1 1 = $-1 + x - x^2 + x^3$
Line 6:     PQ(x) = $p(x) \times q(x)$ # using FFT, PQ(x) polynomial degree of $n + m - 2$
Line 7:     For every coefficient $c$ and exponential value $e$ in PQ(x):
Line 8:         If $m - 1 \leqslant e \leqslant n - 1$:
Line 9:             $d = \frac{m-c}{2}$
Line 10:            If $d \leqslant k$: i = $e-(m-1)$, incides.add(i)
Line 11:    If len($indices > 0$) : print "Yes", return incides
Line 12:    Else: print "No"
```

Running time:
Let T(n) be the run time of the algorithm with size of n (which is number bits of s).
At line 2, change all bits 0 to -1 of string t will take $\Theta(n)$.
At line 3, reverse s and change all bits 0 to -1 will take $\Theta(m)$.
At line 4, construct polynomial degree of m-1 will take $\Theta(m)$.
At line 5, construct polynomial degree of n-1 will take $\Theta(n)$.
At line 6, doing FFT of PQ(x) will take $\Theta(n\log(n))$ since $m < n$.
The loop from line 7 to 10, PQ(x) has degree of $n+m-2$, iterate through the loop will take $\Theta(n+m)$.
At line 11, and 12 will take constant time $\Theta(1)$ to check and return.
Therefore, the run time of the algorithm would be:

$$\Theta(n)+\Theta(m)+\Theta(m)+\Theta(n)+\Theta(n\log(n))+\Theta(n+m)+\Theta(1)$$

Since $n > m$, Therefore:

$$\boxed{T(n)=\Theta(n\log(n))}$$

Proof of correctness:
During the FFT process, the terms that have the same exponential value will be sum up ($c_0x^3+c_1x^3+\cdots c_kx3=cx^3$). Base on the property, we have an invariant.
_ Invariant: $PQ(x)=p(x)q(x)$. At every step (also index of t) i ($0 \leqslant i \leqslant n-m$ and), all terms that have the same exponential value e will be sum up to $m-2d$. We have a value d which is the number bits differ between string s(size of m) and substring (size of m) of t start at index i ($e=m-1+i$)
_ Base case: When $i=0$, we know that each of bit in s string will be multiply with each of bit in t at index i string which is $c_0=s_0t_{m-1}+s_1t_{m-2}+\cdots+s_{m-1}t0$. We know that if two bits are different value, their product will be value of -1 ($-1\cdot 1=-1$, and d is count when the product is -1. Hence, 2d is the total differ bits in s and substring of t. Therefore,

if they're all match, $c_0 = m$, otherwise $c_0 = m - 2d$. $\Rightarrow$ The Invariant is true for base case.

_ Induction Step: Assume that the Invariant is true at k step, we prove it's also true for k+1 step.
At k step (also at index k of t), we know that the coefficient $c_k = m - 2d(k)$, $d(k)$ is the differ between bits string s $(s_0, \cdots, s_{m-1})$ and bits string t $(t_k, t_{k+1}, \cdots, t_{k+(m-1)})$.
At k +1 step (also at index $k+1$ of t),there will be the sum of product which is $(c_{k+1} = s_0 t_{k+1} + s_1 t_{k+2} + \cdots + s_{m-1} t_{k+1+(m-1)})$, and if bit z $(0 \leqslant z \leqslant)$ in s and $k+1+z$ in t $(k+1 \leqslant k+1+z \leqslant k+1+(m-1))$ are different, $s_z t^{k+1+z} = -1$ (because one them of has to equal to 1, and other one is 0 which is value of -1). If there are $d(k+1)$ differ bits in s and substring of t, 2d is the total differ bits in s and substring of t. Therefore, if they're all match, $c_{k+1} = m$, otherwise $c_{k+1} = m - 2d(k+1)$. Therefore, it's true for $k+1$, it's also true for all k $(0 \leqslant k \leqslant n - m)$.

_ Prove correctness property:

In the algorithm at line 6, we know that all the coefficients of $PQ(x)$ will be $m - 2d(i)$ ($d(i)$ is the differ bits between $s^0, s^1, \cdots, s^{m-1}$ and $t^i, t^{i+1}, \cdots, t^{i+(m-1)}$, and we also know the exponential value $e = m-1+i$ in $PQ(x)$, so $i = m - 1$.
At line 7, we iterate through the polynomial $PQ(x)$, since we check valid exponential value and calculate d and check if $d \leqslant k$ and $i = e - (m-1)$, if it's true, and indice array will append i into the list.
After the loop is terminate, the algorithm will check for the length of the array *indices*. If the length is more than 0, we know that s occurs as a substring of t, with less than or equals to k errors, so it will print "Yes" and return the array *indices*. Otherwise, it prints "No".

(d) **Explanation:**
It's kinda similar to part(c), but in the problem we matching letter instead of bits string. Well, thanks for the hint.
Here is how I would do it:
First, encode each letter(A,C,G,T) into 4 distinct bits string, such that A,C,G,T bits string have only 2 number of bits that differ to each other.
Example: choose A = 0111, C= 1110, G= 1101, T=1011

so diff(A,C) = diff(A,G) = diff(A,T)= diff(C,G) = diff(C,T)= diff(G,T) = 2 bits different.
Now m is number of s times 4, n is number of t times 4.
we change all bits 0 to -1 value of string t and construct polynomial q(x) degree of $n-1$ from it.
Same for string s, plus that we reverse string s, then change all bits 0 to -1 value of string s, and construct polynomial p(x) degree of $m-1$ from it.
Using FFT to find the product of p(x)q(x) which take $\Theta(n\log(n))$ since $m<n$. and then look through all the coefficient of p(x)q(x) and exponential value of x correspond to the coefficient. We want to make sure that these coefficient in form of $m-d$ (because differ bits between 2 letters are 2, but we want the differ of letter, therefore, $d=\frac{d}{2}$) and $d \leqslant k$, and these exponential value in form of $m-1+i$ and i is the valid index.

Here is how to check for valid coefficient and index:
Let $e=$ exponential value, where x^e and $e=m-1+i$, i is the index of t. Since each letter is encoded into 4 bits string, and we start compare the letter at the index that is divisible by 4 (0,4,8,...).Therefore i must be divisible by 4 (i%4 = 0) (1) Here how to find i:

$$e=m-1+i$$

$$i=e-(m-1)$$

$$\boxed{i\%4=0}\ (1)$$

We know that $0\leqslant i\leqslant n-m$ since the last invalid i must go from i to $i+m$. Hence,

$$0\leqslant i\leqslant n-m$$

$$0+m-1\leqslant i+m-1\leqslant n-m+m-1$$

$$m-1\leqslant m-1+i\leqslant n-1$$

$$\boxed{m-1\leqslant e\leqslant n-1}\ (2)$$

Let $c=$ coefficient value of $x^{(m-1+i)}$ in p(x)q(x), we know that

$$c=m-d(i),\ where\ d(i)\ is\ number\ of\ bits\ differ$$

$$d(i) = m - c$$

$$\boxed{d(i) \leqslant k} \; (3)$$

If all (1) and (2) and (3) are true. We add i to the list of indices that we want to return.
Finally, check for the list of indices, if list is not empty, we know that s occurs as a substring of t with $\leqslant k$ errors at some indices, so we just output "Yes" and return the list of indices. Otherwise, the list if empty, and we output "No".

Algorithm:

```
Line 0: Matching(DNA s, DNA t, k)
Line 1:     m = number letter of s times 4, n = number letter of t times 4
Line 2:     indices = empty array
Line 3:     $t\prime$ = encode each letter of t into 4 bits and change all bits 0 to -1 of string t
Line 4:     $s\prime$ = encode each letter of s into 4 bits reverse s and change all bits 0 to -1
Line 5:     $p(x)$ = construct polynomial degree of $m-1$ from $s\prime$ # 1 1 -1 = $1 + x - x^2$
Line 6:     $q(x)$ = construct polynomial degree of $n-1$ from $t\prime$ # -1 1 -1 1 = $-1 + x - x^2 + x^3$
Line 7:     PQ(x) = $p(x) \times q(x)$ # using FFT, PQ(x) polynomial degree of $n + m - 2$
Line 8:     For every coefficient $c$ and exponential value $e$ in PQ(x):
Line 9:         If $m - 1 \leqslant e \leqslant n - 1$:
Line 10:            i = $e - (m - 1)$
Line 11:            If i% 4 = 0:
Line 12:                $d = m - c$
Line 13:                If $d \leqslant k$: incides.add(i)
Line 14:    If len($indices > 0$) : print "Yes", return incides
Line 15:    Else: print "No"
```

Running time: Let T(n) be the run time of the algorithm with size of n (which is number bits of s).
At line 3, encode and change all bits 0 to -1 of string t will take $\Theta(n)$.
At line 4, encode and reverse s and change all bits 0 to -1 will take $\Theta(m)$.
At line 5, construct polynomial degree of m-1 will take $\Theta(m)$.
At line 6, construct polynomial degree of n-1 will take $\Theta(n)$.
At line 7, doing FFT of PQ(x) will take $\Theta(n \log(n))$ since $m < n$.
The loop from line 8 to line 13, PQ(x) has degree of $4n + 4m - 2$, iterate through the loop will take $\Theta(n + m)$.
At line 14, and 15 will take constant time $\Theta(1)$ to check and return.

Therefore, the run time of the algorithm would be:

$$\Theta(n) + \Theta(m) + \Theta(m) + \Theta(n) + \Theta(n\log(n)) + \Theta(n+m) + \Theta(1)$$

Since $n > m$, Therefore:

$$\boxed{T(n) = \Theta(n\log(n))}$$

Proof of correctness:
During the FFT process, the terms that have the same exponential value will be sum up ($c_0x^3 + c_1x^3 + \cdots c_kx3 = cx^3$).Also we encode A = 0111, C= 1110, G= 1101, T=1011, and number of differ bits between any of these two letters are 2. Since we encode each letter into 4 bits and different bits between 2 letter (not the same letter) is 2 bit, so instead of $m - 2d$ (d is the number of differ bits between 2 bits string), the coefficient value of $PQ(x)$ will be in form of $m - d$ (because we only count the different in letters, not bits).
Base on the property, we have an invariant.
_ Invariant: $PQ(x) = p(x)q(x)$. At every step (also index of t) i ($0 \leqslant i \leqslant n - m$ and), all terms that have the same exponential value e will be sum up to $m - d$. We have a value d which is the number bits differ between string s(size of m) and substring (size of m) of t start at index i ($e = m - 1 + i$)

_ Base case: When $i = 0$, we know that each of bit in s string will be multiply with each of bit in t at index i string which is $c_0 = s_0t_{m-1} + s_1t_{m-2} + \cdots + s_{m-1}t0$. We know that if two bits are different value, their product will be value of -1 ($-1 \cdot 1 = -1$, and d is count when the product is -1. Hence, 2d is the total differ bits in s and substring of t. Therefore, if they're all match, $c_0 = m$, otherwise $c_0 = m - d$. $\Rightarrow$ The Invariant is true for base case.

_ Induction Step: Assume that the Invariant is true at k step, we prove it's also true for k+1 step.
At k step (also at index k of t), we know that the coefficient $c_k = m - d(k)$, $d(k)$ is the differ between bits string s ($s_0, \cdots, s_{m-1}$) and bits string t ($t_k, t_{k+1}, \cdots, t_{k+(m-1)}$).
At k +1 step (also at index $k + 1$ of t),there will be the sum of product which is ($c_{k+1} = s_0t_{k+1} + s_1t_{k+2} + \cdots + s_{m-1}t_{k+1+(m-1)}$), and if bit z

$(0 \leqslant z \leqslant)$ in s and $k+1+z$ in t $(k+1 \leqslant k+1+z \leqslant k+1+(m-1))$ are different, $s_z t^{k+1+z} = -1$ (because one them of has to equal to 1, and other one is 0 which is value of -1). If there are $d(k+1)$ differ bits in s and substring of t, d is the total differ 2 bits (2 bits differ $\rightarrow$ count 1) in s and substring of t. Therefore, if they're all match, $c_{k+1} = m$, otherwise $c_{k+1} = m - d(k+1)$. Therefore, it's true for $k+1$, it's also true for all k $(0 \leqslant k \leqslant n-m)$.

_ Prove correctness property:

In the algorithm at line 7, we know that all the coefficients of $PQ(x)$ will be $m - d(i)$ ($d(i)$ is the differ bits between $s^0, s^1, \cdots, s^{m-1}$ and $t^i, t^{i+1}, \cdots, t^{i+(m-1)}$, and we also know the exponential value $e = m-1+i$ in $PQ(x)$, so $i = m-1$.

At line 8, we iterate through the polynomial $PQ(x)$, since we check valid exponential value and calculate $i = e - (m-1)$,check $i\%4 = 0$, calculate d and check if $d \leqslant k$, if it's true, and indice array will append i into the list.

After the loop is terminate, the algorithm will check for the length of the array *indices*. If the length is more than 0, we know that s occurs as a sub-letter of t, with less than or equals to k errors, so it will print "Yes" and return the array *indices*. Otherwise, it prints "No".

6. Triple sum

Explanation: Thanks for the hint.
At first I construct a polynomial will all coefficients are value of 1, and exponential values are the value of elements in array A. Since we know that every element in array A is $0 \leqslant A[i] \leqslant 10n$, we know that the most degree of the polynomial will be $10n - 1$.
Then, we apply FFT twice to the polynomial to raise to the power of 3. We know that after apply FFT twice, all the exponential values in that polynomial are all the combination of triple sum of all elements in array A.
Finally, all we have to do is iterate through the polynomial which we apply FFT twice, and get the exponential value, compare it with t. If it's the same as t , it means that there are 3 elements in array A that add up to t, and print "Yes". Otherwise, print "No".

Algorithm:

```
Line 0: TripleSum(A[0,$\cdots$, $n-1$]):
Line 1:     p(x) = construct a polynomial with all coefficient is 1 and exponential values are element from A # 1 2 3 4 = $x + x^2 + x^3 + x^4$
Line 2:     q(x) = $(p(x))^3$ # apply FFT twice
Line 3:     For each exponential value c in q(x):
Line 4:         if (c = t): print "Yes", return
Line 5:     print "No"
```

Running time: Let T(n) be the run time of the algorithm TripleSum with size of n elements in array A.
At line 1, construct a polynomial will take $\Theta(n)$
At line 2, apply FFT 1st time take $\Theta(n \log(n))$ since p(x) is polynomial degree of (n-1) with p(x) also degree (n-1). After the 1st FFT, the most degree of the $(p(x))^2$ would be $(10n-1)+(10n-1) = 20n-2$. Second time apply FFT will take $\Theta(n \log(n))$ because $(p(x))^2$ has most degree of $20n-1$ and p(x) has most degree of $10n-1$, so apply FFT will take less than $(20n-2)\log(20n-2)$ which is $\Theta(n \log(n))$.
The loop from line 3 to 4, we iterate through exponential value c in q(x) till if there is any $c = t$, if there is no solution, it will takes $\Theta(n)$ (assume there

are 30n element in $q(x)^3$).
Therefore, the run time would be:

$$\Theta(n) + \Theta(n\log(n)) + \Theta(n\log(n)) + \Theta(n)$$

$$\boxed{T(n) = \Theta(n\log(n))}$$

Proof of correctness:
_ Induction Hypothesis:
If P(x) is a polynomial of degree $n-1$,and all coefficients of value 1.

$$P(x) = x^0 + x^1 + \cdots x^{n-1}$$

At some i such that $0 \leqslant i \leqslant n-1$, if doing $P(x)^3$, x^i will multiply (including x^i) with every single x^j and x^z in $P(x)$ such that $0 \leqslant j, z \leqslant n-1$. Therefore, $P(x)$ will have the exponential value of $i+j+k$ for which x^{i+j+k}

_ Base Case: At first doing $P(x)^2$, x^0 will multiply with every single x^i such that $0 \leqslant i \leqslant n-1$, so we have x^{0+i}. Doing $P(x)^3$, $x0+i$ will multiply with every single x^j such that $0 \leqslant j \leqslant n-1$, so now we have x^{0+i+j}. Therefore, the induction hypothesis is true for the base case.

_ Induction Step: Assume that the Invariant is true at some value k exponential.

$$P(x) = x^0 + \cdots + x^k + \cdots + x^{n-1}$$

Doing $P(x)^3$, we know that there would be a exponential value x^{k+i+j} $0 \leqslant j \leqslant n-1$.

$$P(x) = x^0 + \cdots + x^k + x^{k+1} \cdots + x^{n-1}$$

Doing $P(x)^2$, we can see that x^{k+1} will multiply with every single $x^j = i$ such that $0 \leqslant i \leqslant n-1$, so we have x^{k+1+i}. Doing $P(x)^3$, $xk+1+i$ will multiply with every single x^j such that $0 \leqslant j \leqslant n-1$, so now we have $x^{k+1+i+j}$. Therefore, it's true for $k+1$, so it's true for all k ($0 \leqslant k \leqslant n-1$). Hence, the Hypothesis in inductive.

_ Contradiction proof:
Given that there is an array $A[0 \cdots n-1]$ and elements in A is in the range

$0 \leqslant A[i] \leqslant 10n$, and there exist indices i,j,k such that $A[i] + A[j] + A[k] = t$.

Assume that t is the triple sum of 3 elements in array A (could be including the same element), and the algorithm print "No" for which the algorithm cannot find t equals to triple sum of 3 elements in array A (could be including the same element).

At line 3: We know that doing $q(x) = (p(x))^3$ by apply FFT twice will give us all the triple sum combination of all the exponential in $p(x)$.
Hence, at the line 3 and 4, by iterate through the exponential values called c in $q(x)$, and we know c is the triple sum of some 3 elements in array A (could be include the same element), and we also know that t is the triple sum of some 3 elements in array A (could be include the same element). Therefore, c must equal to t, and the algorithm will print "Yes", and return which is terminate the program. However, there is a contradiction, we said that the algorithm print "No". Therefore, the algorithm can find the triple sum solution.

7. Optional bonus problem: More medians

Explanation:
Algorithm: YOUR ANSWER GOES HERE
Running time: YOUR ANSWER GOES HERE
Proof of correctness: YOUR ANSWER GOES HERE