I certify that the work submitted with this exam is mine and was generated in a manner consistent with this document, the course academic policy on the course website, and the Rutgers academic code. I also certify that I will not disclose this exam to others who are taking this course and who will take this course in the future without the authorization of the instructor.

Date: __10/28/2022______________________

Name (please print): ____Dong Shu____________________

Signature: _____Dong Shu____________________

Name: ________Dong Shu________________________ Section: __________03________________

Problem Number(s)	**Possible Points**	**Earned Points**
1	10	
2	10	
3	10	
4	10	
5	10	
6	15	
7	5	
8	10	
9	10	
10	10	
Total	100	

Exam Time: 12 hours, 10 problems (14 pages, including this page)

- Write your name on this page and the last page, put your initials on the rest of the pages.
- If needed, use the last page to write your answer.
- Show your work to get partial credits.
- Show your rational if asked. Just giving an answer can't give you full credits.
- You may use any algorithms (procedures) that we learned in the class.
- Keep the answers as brief and clear as possible.

Name: ________Dong Shu________________________ Section: _________03________________

[Note: In this exam, both lg(x) and log(x) means $\log_2(x)$]
[Math facts: $\log a^b = b\log a$, $a^{\log b} = b^{\log a}$]

1. **(10 points) Asymptotic Growth of Functions**

List the 5 functions below in non-decreasing asymptotic order of growth:

$(\log n)^2$ $\log\log n$ n $n\log n$ $\log n$

(1) __loglogn____ (2) ____logn___ (3) ____(logn)^2____ (4) ____n____ (5) ____nlogn___
smallest largest

2. **(10 points) Properties of O, Ω, Θ**

Clues:

(1) $f_1(n) = \Omega(n^2)$ (2) $f_2(n) = O(\log n)$ (3) $f_3(n) = \Theta(n)$
(4) $f_4(n) = O(2^n)$ (5) $f_5(n) = \Omega(n!)$

Circle TRUE (the statement must be always TRUE based on the clues above) or circle FALSE otherwise.

(a) $f_1(n) = \Omega(f_2(n))$ TRUE FALSE

(b) $f_1(n) = O(f_5(n))$ TRUE FALSE

(c) $f_2(n) = O(f_3(n))$ TRUE FALSE

(d) $f_3(n) = O(f_4(n))$ TRUE FALSE

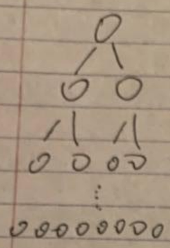
(e) $f_5(n) = \Omega(f_4(n))$ TRUE FALSE

Name: ________Dong Shu_________________________ Section: __________03__________________

3. **(10 points) Recursion Trees**
Use the recursion tree method to determine the asymptotic upper bound of $T(n)$.
$T(n)$ satisfies the recurrence $T(n) = 2T(n-1) + n$, and $T(0)=0$.

Name: ________Dong Shu________________________ **Section:** __________03________________

3. $T(n) = 2T(n-1) + n$



1 problem of size n

2 problem of size $n-1$

2^k problem of size $n-k$

2^n problem of size 0

$$T(n) = n + 2\cdot(n-1) + 2^2(n-2) + \cdots + 2^{n-1}\overset{n-(n-1)}{(1)} + 2^n\overset{n-n}{(0)}$$

$$= n + 2n + 4n + \cdots + 2^{n-1}(n) + 2^n n - 2 - 2^2 2 - \cdots - 2^{n-1}(n-1) - 2^n(n)$$

$$= 2^{n-1} + \underbrace{\sum_{k=0}^{n-2} 2^k(n-k)}$$

$$= n\sum_{k=0}^{n-2} 2^k - \sum_{k=0}^{n-2} k2^k$$

$$= n(2^{n-1}-1) - \frac{n\cdot 2^n - 3\cdot 2^n + 4}{2}$$

$$= n(2^{n-1}-1) - (n\cdot 2^{n-1} - 3\cdot 2^{n-1} + 2)$$

$$= 3\cdot 2^{n-1} - n - 2$$

$$T(n) = 2^{n-1} + 3\cdot 2^{n-1} - n - 2$$

$$= 4(2^{n-1}) - n - 2$$

$$= 2^{n+1} - n - 2$$

$$= O(2^n)$$

Name: ________Dong Shu_________________________ Section: __________03_________________

4. **(10 points) Solving Recurrences**

(1) (5 points) Find a tight bound solution for the following recurrence:

$$T(n) = 4\,T(\tfrac{n}{2}) + c\,n \qquad \text{(c is a positive constant)}$$

That is, find a function g(n) such that $T(n) \in \Theta(g(n))$. For convenience, you may assume that n is a power of 2, i.e., $n=2^k$ for some positive integer k. Justify your answer. [Note: Read question 4-(2) first before writing your answer]

We can first use master theorem and then use iteration method to prove it.

For $T(n) = 4\,T(\frac{n}{2}) + c\,n$, we have a = 4, b = 2, d = 1. In this case, a > b^d, because 4 > 2^1. So we have T(n) ∊ Θ(n^(log_b a)) = Θ(n^(log_2 4)) = Θ(n^2)

Name: ________Dong Shu________________________ Section: _________03_______________

(2) (5 points) Prove your answer in 4-(1) either using the iteration method or using the substitution method that we learned in our class.

Name: ________Dong Shu_________________________ Section: __________03_________________

4.b $T(n) = 4T(n/2) + cn$

$T(n/2) = 4T((n/2)/2) + cn/2$

$= 4T(n/4) + cn/2$

substitude:

$T(n) = 4(4T(n/4) + cn/2) + cn$

$= 4^2 T(n/4) + 3cn$

$T(n/4) = 4T(n/8) + cn/4$

substitude

$T(n) = 4^2(4T(n/8) + cn/4) + 3cn$

$= 4^3 T(n/8) + 7cn$

∴ eventually we have:

$T(n) = 4^k T(n/2^k) + (2^k - 1)cn$

we have $2^k = n$ $k = \log n$

$= 4^{\log n} T(n/2^{\log n}) + (2^{\log n} - 1)cn$

$= 4^{\log n} T(1) + (n-1)cn$

$= n^2 T(1) + (n-1)cn$

Because $T(1) \le c$

$T(n) = cn^2 + (n-1)cn$

$= \Theta(n^2)$

Name: ________Dong Shu________________________ Section: _________03________________

5. **(10 points) Complexity of Sorting Algorithms**
   Sort a number of *n* integers. The value of the integers ranges from 0 to n^4-1. Please provide the worst-case running time for the following sorting algorithms (1) ~(5) provided in our lecture, using big-O or big-Θ notation wherever appropriate.

   (1) (1 point) Insertion Sort:
O(n^2)

   (2) (1 point) Merge Sort:
   Θ (nlogn)

   (3) (1 point) Quick Sort:
   O(n^2)

   (4) (1 point) Counting Sort:
O(n+k), and we know that k <= n^4-1, so O(n^4)

   (5) (1 point) Radix Sort:
   O(d(n+k)), and we know that value of the integers <= n^4-1, and k is the range of each digit, so k is at most 10. We do not know the value for d, since n^4-1 can have lots of digits. Therefore, if our number of integers n is greater than 10, then the worst case will be O(dn), if not then it will be(dk).

   (6) (5 points) What is the average-case running time for Merge Sort and Quick Sort, respectively? Why do we prefer Quick Sort algorithm in practice? Please provide at least two benefits of Quick Sort and also provide explanations for each benefit (plain language explanation is fine).

   The average case running time for Merge Sort is O (nlogn)
   The average case running time for Quick Sort is O(nlogn)

   The first benefit of Quicksort requires little space. That means it is a smarter manipulation method that requires no extra memory storage for left sub-array and right sub-array after we select the pivot, because we can separate the array in place.

Name: ________Dong Shu________________________ Section: _________03________________

The second benefit of Quick sort is that it is easy to avoid worst-case run time of O(n^2) almost entirely by using randomly selection. Because it is very difficult that we happen to select the smallest or biggest element every time when we pick a pivot.

6. **(15 points) Stability of Sorting Algorithms**
(1) (5 points) If a sorting algorithm is *stable*, what does it mean? Explain it clearly.

The algorithm is stable if two equal key objects maintain the same order in sorted output as the input.
For example, if we have an array of the same element, 1 1 1 1 1
After we use a stable sorting algorithm, the first 1 will still remain at the original position, and the second 1 will still remain at the original position, and so on.

(2) (5 points) Consider sorting an unsorted array A using the algorithms provided in the lecture, are merge sort and quick sort stable? Circle your answer below

Insertion Sort	Stable	Unstable
Merge Sort	Stable	Unstable
Quick Sort	Stable	Unstable
Counting Sort	Stable	Unstable
Radix Sort	Stable	Unstable

Name: ________Dong Shu________________________ Section: _________03________________

(3) (5 points) In Radix Sort, why do we need to use a stable sorting algorithm to sort the numbers in each bucket? Provide a clear explanation.

Because we are first sorting the right most digit of each element, if we do not use stable sorting algorithm we will lose the sequence of the digit in right position. For example, if we have 92, 90, 97 in our array, and we first sort the right most digit, then we will get 90, 92, 97. However, if we do not use stable sorting algorithm to remember the order of previous sort, then when we do next sort, it is possible we end up with 92, 90, 97 or 97, 90, 92 and so on, because the next sort digit are all the same. If we use stable sorting algorithm, then we will remember the order, and we will successfully have 90, 92, 97.

7. **(5 points) Quick Sort and Algorithm Analysis**
We have learned in class that while the expected running time of the randomized version of quicksort is O($n$log$n$), the worst-case running time is O(n^2). Show how quicksort can be made to run in O($n$log$n$) time in the worst case. Assume the input array is A[0:n-1] and all elements in A are distinct. Write your answer as pseudo-code and use plain language to explain the idea of your algorithm.
(Hint: you can use any algorithm we have learned in the class as helper functions in parts of your designed algorithm. If you use an algorithm we learned in class as a helper function, you can directly call the function name in your pseudo-code without expanding the details of the helper function, as long as you clearly explain the meaning of the helper function.)

The reason why we will have worst case running time of O(n^2) is because that we happen to choose the largest number or smallest number as our pivot in every run. In this case, we can improve our algorithm by using smartly_choose_pivot and select_k method, so that we can smartly choose our pivot and avoid to choose the largest number or smallest number. Since our smartly_choose_pivot worst case running time is O(n), so that our total running time will be O(nlogn + n) = O(nlogn)

Name: ________Dong Shu_________________________ Section: __________03_________________

```
Algorithm quicksort(A):
        If length(A) <= 1:
                Return
        p = smartly_choose_pivot(A)
        L, A[p], R = partition(A, p)
        quicksort(L)
        quicksort(R)
```

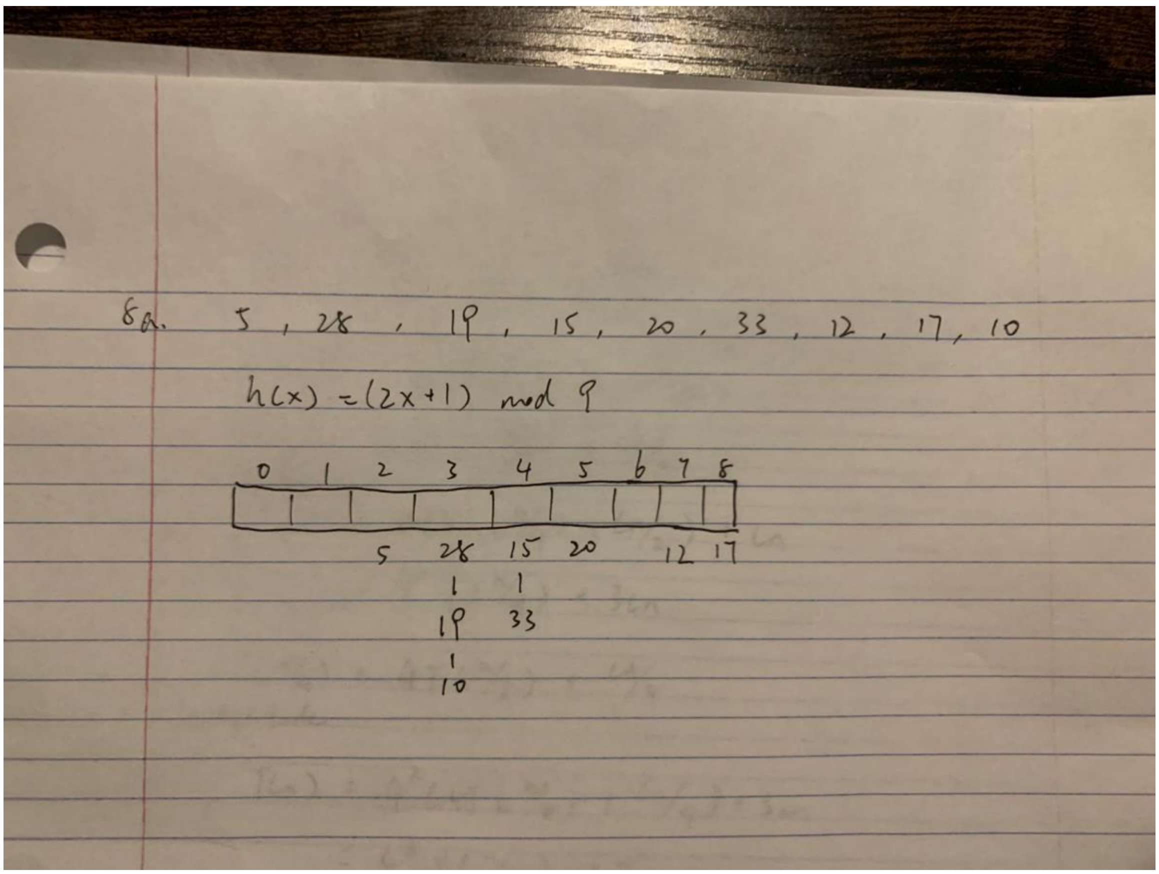
```
Algorithm smartly_choose_pivot(A):

        groups = split A into m=⌈length(A)/5⌉
                    groups, of size ≤ 5 each
        candidate_pivots = []
        for i = 0 to m-1:
                p_i = median(groups[i])  # O(1)
                candidate_pivots.append(p_i)
        A[p] = select_k(candidate_pivots, m/2)
        return index_of(A[p])
```

Name: ________Dong Shu________________________ **Section:** _________03________________

8. **(10 points) Randomized Algorithms and Hashing**

(1) (5 points) Show the result when we insert the keys 5; 28; 19; 15; 20; 33; 12; 17; 10 into a hash table with collisions resolved by linked list at each slot. Let the hash table have 9 slots, and let the hash function be *h(x) = (2x+1) mod 9*. (You are expected to draw the final hash table)



Name: ________Dong Shu_________________________ Section: __________03_________________

(2) (5 points) Suppose we hash elements of a set U of keys into m slots. Show that if $|U| > (n - 1)m$, then there are at least n keys that all hash to the same slot, so that the worst-case searching time for hashing with linked list to resolve collisions is $\Theta(n)$.

Since our |U| is greater than (n-1)m, so let us suppose that |U| >= (n-1)m + 1. We also know that there are m slots, and there are at most (n-1) keys in each slot, because there are at least (n-1)m keys. In this case, when we add one more element from |U| >= (n-1)m + 1, we have to add that element to the slot where it already has n-1 keys. In this case, one of our slot will have n keys, which result to resolve collisions of $\Theta(n)$.

9. **(10 points) Probability of Collision**

A frequently used hash family is the matrix multiplication hash family that we have introduced in class.

Suppose we have n buckets {$1, 2, \ldots, n$}, we will use a binary string of length $b = \log_2 n$ to index each bucket. (For example, if we have 4 buckets, they will be indexed as 00, 01, 10, 11.)

Now, we would like to hash a u-bit binary string x into the hash table (For example, x could be an 8-bit string 10100110). To hash this u-bit string x into a

Name: ________Dong Shu________________________ Section: _________03________________

bucket, we use the hash function $h_A(x) = (Ax) mod\ 2$, where A is a $b \times u$ dimensional binary matrix, and x is a $u \times 1$ dimensional column vector. As a result, $h_A(x)$ will be a $b \times 1$ dimensional vector which shows the bucket index that x will be hashed to. (For example, x is an 8-bit string $[10100110]^T$, and A is a 2×8 dimensional matrix $\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$, then $h_A(x) = (Ax) mod\ 2 = [1, 0]^T$, which means that x will be hashed to bucket 2.)

Prove that the above hash function $h_A(x) = (Ax) mod\ 2$ is a <u>universal hash family</u>. [Hint: Consider hashing two arbitrary keys x and y into the hash table, both x and y are u-bit binary strings.]

We know that in order to be a universal hash family, the probability of a collision have to upper bounded by 1/n

The function for the probability of a collision is hA(x) − hA(y) = 0, and we know that $h_A(x) = (Ax) mod\ 2$. Therefore, our function can be write like this:

hA(x) − hA(y) = 0
A(x) mod 2 – A(y) mod 2 = 0
P[A(x-y) mod 2] = 0

We know that A is a b * u matrix, and (x – y) is a u * 1 matrix. As the result of these two matrices, A(x-y) will be a b*1 matrix.

So P[A(x-y) mod 2] = 0 will become:

b*1 matrix mod 2 = 0
Finally we will have:
Answer b*1 matrix = 0

In order to make our answer b*1 matrix = 0, our elements in answer b*1 matrix need all be 0. In order to make that happen, our pervious b*1 matrix before mod 2 should only contain even number. Since our number in matrix A are arbitrary number, so our b*1 matrix will also have arbitrary number. The probability of one arbitrary element is even number is ½, and there are b arbitrary elements in the matrix. Therefore, our probability of all elements in b*1 matrix are even numbers is 1/2^b. Since we already know that b = log_2 n. So the probability of collision will be 1/2^b = 1/2^log_2 n = 1/n, which is upper bounded by 1/n
Therefore, we proved that function $h_A(x) = (Ax) mod\ 2$ is a universal hash family.

Name: ________Dong Shu_________________________ Section: __________03_________________

10. **(10 points) Design an Algorithm**

You are given an array *A,* which stores *n* non-negative integers. Design an *efficient* **divide-and-conquer** algorithm that accepts *A* and *n* as inputs and returns the **index of the maximum value** in the array *A*.

(1) (5 points) **Basic idea and Pseudocode**: *(Pleas first use a few sentences of plan language to explain the basic idea of your algorithm, then provide the pseudo-code of your algorithm. Please be clear about each step of your pseudo-code, when necessary, you can add comments to explain each step)*

In this algorithm, if length of array smaller or equal to 2, then I will just compare those. If length of array greater than 2, then I will separate the array into Sub left and Sub Right. Then, I will find the maximum number in the sub left part and sub right part. After that I will compare the maximum number from two sub part and return the index of the maximum number of the array.

```
Algorithm MaxVal(A):
    If A.length <= 1:
            Return
    Else If A.length = 2:
            Compare two indexes
            Return indexOf(Maximum)
    Else:
            Sub_left = left half of the array
            Sub_right = right half of the array
            maxLeft = MaxVal(Sub_left)
            maxRight = MaxVal(Sub_right)
            Compare maxLeft and maxRight
            Return indexOf(maximum)
```

Name: ________Dong Shu_________________________ Section: __________03__________________

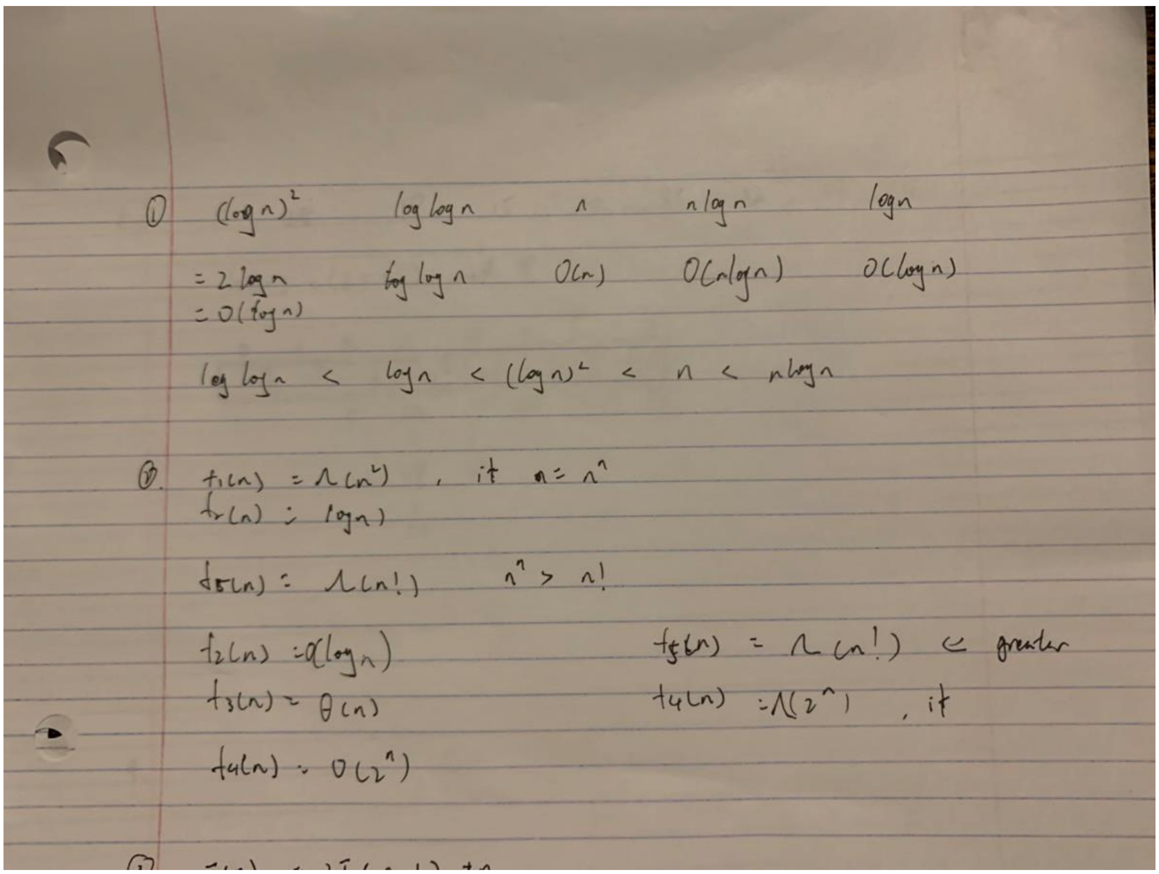
(2) (5 points) **Analysis:** derive a recurrence for the running time T(n) of your algorithm, and solve the recurrence to provide the asymptotic running time of your algorithm.

When we have less or equal 2 elements, T(n) will be T(1).
When we have more than 2 elements,
T(n) = 2T(n/2) + c
We can use master theorem
we have a = 2, b = 2, d = 0
a > b^d
T(n) = O(n^(log_b a)) = O(n^(log_2 2)) = O(n)

Name: ________Dong Shu________________________ Section: _________03________________

(You may use this page to write answers if needed. Please mark the problem number clearly)

Some background calculation steps for different questions. **Not the answer!**



① $(\log n)^2$ $\quad \log\log n$ $\quad n$ $\quad n\log n$ $\quad \log n$

$= 2\log n$ $\quad \log\log n$ $\quad O(n)$ $\quad O(n\log n)$ $\quad O(\log n)$

$= O(\log n)$

$\log\log n < \log n < (\log n)^2 < n < n\log n$

②. $f_1(n) = \Omega(n^2)$, if $n = n^n$

$f_2(n) : \log n)$

$f_5(n) : \Omega(n!)$ $\quad n^n > n!$

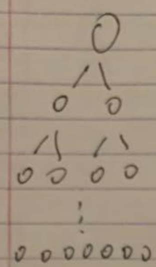
$f_2(n) = O(\log n)$ $\qquad f_5(n) = \Omega(n!) \Leftarrow$ greater

$f_3(n) = \theta(n)$ $\qquad f_4(n) = \Omega(2^n)$, if

$f_4(n) = O(2^n)$

Name: ________Dong Shu________________________ **Section:** __________03________________

3. $T(n) = 2T(n-1) + n$



1 problem of size n

2 problem of size $n-1$

2^k problem of size $n-k$

2^n problem of size 0

$$n + 2\cdot(n-1) + 2^2(n-2) + \dots + 2^{n-1}(\underset{n-(n-1)}{1}) + 2^{(n)}(\underset{n-n}{0})$$

$$= n + 2n + 4n + \dots + 2^{n-1}(n) + 2^n n - 2 - 2^2\cdot 2 - \dots - 2^{n-1}\cdot(n-n-1) - 2^n\cdot(n-n)$$

assume this is $f(x)$, $\therefore 2f(x) - f(x) = f(x) =$

$$= 2^{n+1}n - n - (2\cdot 2 - 2^3\cdot 2 - 2^4\cdot 3 - \dots - 2^n(n-n-1) - 2^{n+1}\cdot(n-n)) - (-2 - 2^2\cdot 2 - 2^3\cdot 3 - \dots - 2^{n-1}(n-n-1) - 2^n(n-n))$$

$$= 2^{n+1}n - n - (\underbrace{2 - 2^2 - 2^3 - 2^4 - \dots - 2^n}_{\text{assume is } f(x)})$$

, $2f(x) - f(x) = f(x)$

$$= 2^{n+1}n - n - (2^{n+1} - 2)$$

$$= (2^{n+1} - 1)n$$