**NEW YORK UNIVERSITY BRIDGE TO TANDON**

**Pre-Program**

*The Secret Rules of Modern Living Algorithms*

* Best algorithms are those that tap into the underlying mathematical structure hiding beneath a problem.
* Euclid’s Algorithm (Oldest Algorithm) is developed to find GCD.
* Google Websearch Algorithm
  + **PageRank**
  + Higher rank to the website that has lot of links from other websites.
* Sorting Algorithms
  + **Bubble sort**: Consider the items in pair and sort between them and move along and repeat.
  + **Merge sort**
    - Jon Von Neumann devised it.
    - Divide and Conquer
  + There are nearly **20** sorting algorithms.
* Matching Algorithm
  + Gale –Shapley Algorithm
  + Dating, Students to schools, universities.
* Importance of an algorithm is **correctness** and **speed**.
* Heuristics: A way of devising algorithms that can solve the problem but not really the efficient one.

———————————————————————————————————————————————————

**Programming Tips**

**Shapes**

* Nested loops (1 for row and 1 for columns).
* 1st loop based on no of rows.
* 2nd loop based on no of columns.
* Draw the diagram for a n, come up with a table, calc for each row #char #spaces and come up with an equation.

**Programming Learnings**

1. Vectors are not pass by reference by default like arrays are.
2. ranged for loop cannot be used with an array given as a function parameter (soln found in recursion\_mod)
3. Pointer to pointers have 2 most common uses
   1. Use ptrptr as a formal parameter in a function to change the value of a pointer in main function from this function.
   2. Dynamically create an array of pointers.
   3. Create multidimensional dynamic arrays

——————————————————  **PROGRAMMING  QUESTIONS** —————————————————

**PART II (W5 - W8)**

1. Program to print the nth element in the fibonacci sequence.
2. Program to
   1. Print a shifted triangle with the given parameters.
   2. Print a pine tree consisting of triangles of increasing sizes.
3. Program to print the year calendar for the given year.
4. Program to print all the divisors of the given number (given T(n))
5. Program to print all the perfect and amicable numbers within the given range
6. Program to print the euler's number approximated to the given number of terms.
7. Program to print the minimum value from a sequence and all it's indices.
8. Program to check if the given word is a palindrome or not.
9. Program to 1) reverse the array, 2) remove Odd numbers, 3) split parity,
10. Program to simulate randomized challenge-response system (PIN)
11. Program to print a person’s name in the given format.
12. Program to replace digits with x in a line of text.
13. Program to read a line of text and output number of words in the line
14. Program to check if two given strings are anagrams.
15. Program to reorder the elements of an array so that all odd numbers come first in the same order and evens will be placed after odd in a reversed order.
16. Program to create an array with all words in a sentence as elements.
17. Program to get the positive numbers from an array using 4 different implementations (Arrays, ptrs with and without return)
18. Program to search a number in a sequence of positive integers and print their lines.
19. Program to find the missing numbers in a range.
20. Program that implements recursion to print right triangle, opposite triangles and rulers.
21. Program that implements recursion to a) print sum of squares and b) find if the elements in an array are sorted in asc order
22. Program that implements recursion to 1) find the min value from all the elements in an array and 2) find the min value within the given positions in the array
23. Program that uses recursion to implement jump board game.

**PART III (W9-W12)**

——————————————————  **PROGRAMMING  QUESTIONS** —————————————————

**LECTURES**

**Week 1 Lecture**

1. 2’s Complement I
   1. It’s a different no system —> not Base 2 or Binary because Binary uses -1101 format for -ve
      1. (11001100)8 bit 2’C != (11001100)2
   2. Since computers can use only 1 or 0, 2’s C is used to represent -ves in Computers.
   3. Should specify the length of bits in 2’s complement.
   4. ( )8bit 2’s C
   5. 3rd shortcut
      1. Move from right to left.
      2. Keep the 0’s and 1 till you hit the first 1.
      3. Swap the digits from then.
2. Floating point numbers conversion
   1. For decimals take exponents -1,-2,-3, etc and add values
   2. For computers it’s different.

**Week 2 Lecture**

1. Topics in the exam —> Hw is a good indicator.
2. Always declare variable only when you use it (not beforehand).
3. In exams and Hw —> don’t use any features not talked about in the program.

**Week 5 Lecture**

Exam

1. 10 questions for math (+ 2  for programming)
2. For math 1 cpp and for each programming questions 1 cpp file.
3. .cpp files to be submitted for both math and programming
4. Mostly MCQ and few single line answer questions
5. Duration: 2 hours - Own timer. (Get a stop clock)
6. Once you submit the password - Timer starts.
7. Should submit the exam before the 2 hour window.
8. Calculator — IDE

**Week 7 Lecture**

* Data types used while working with pointers
  + int\*
  + double\*
  + float\*
  + char\*
  + int\*\* var —> var stores address of a pointer (var points to a pointer)
* Operators used when working with pointers
  + &var - address of operator: fetches the address of the var it is used with.
  + \*ptr - dereferencing operator: fetches the value of the var ptr is pointing to
* \*(&x) <-> x (\* and & are inverse of one another)
* The memory space allotted for pointer variables (ptr) is 4 bytes (8 bytes for x64bit) irrespective of the DT it’s pointing to.
* Types for pointer are used only as reference to the compiler for the type of the value the ptr is pointing to rather than indicating the size of the pointer variable.
* Things we do with pointers
  + Alternative way to change value of variables out of its active scope.
    - func(&n);  void func (int\* pn) { \*pn = \*pn + 1;}
  + Extend the ways to work with arrays (dynamic arrays).
    - int \*arrPtr = new int[size];
    - delete[] arrPtr; arrPtr = nullptr;
  + Memory allocation.
    - Int \*ptr = new int;
    - delete ptr; ptr = nullptr;
    - Error handling should be done to make sure mem is allocated and null is not returned (in general scenario, but not in this course)

**Week 8 Lecture**

* Recursion is a problem solving technique that’s closely related to MI where we define a solution as a combination of solutions of smaller possible inputs.
* Two cases
  + Base case:
    - Identify how to **measure the input size**
    - Find the **condition** for smallest input
    - Formulate **solution** for smallest input
  + Recursive step:
    - Assume calling the **function** on a **smaller input** does its **job**.
    - Based on assumption, find how to **combine** calls to smaller instances to solve the problem for given larger input.
  + Use the above sentences to design the recursive algorithm step by step.
  + Replace the highlighted part in the sentence with problem specific description.
* Run time Analysis of Recursive Algorithms
  + Recursive Tree
    - Structure
      * Each recursive call is represented as a node in the tree with value as size of the input.
      * Edge represents the direct calls from function (node).
    - Cost
      * At the side of each node, we write the local cost (cost without considering cost of recursive call)
      * If we add all the local costs, we get the total cost of recursive process

**NOTES**

27/08/20

**M12.2 RECURSION**

* Recursion is a problem solving technique similar to mathematical induction.
* Algorithm is developed in 2 steps - Base case, Recursive step.
* Ex1: printAsc(int start, int end);
  + Solution 1: printAsc(start, end - 1); cout << end;
  + Solution 2: cout<< start; printAsc(start+1, end);
  + Solution3: mid = start + end / 2; printAsc(start, mid); printAsc(mid + 1, end);
* Developing a Recursive Algorithm
  + Base case: Solve the problem for the smallest possible input.
  + Recursive case: Define the inductive hypothesis (Call the function with reduced input size to reach the base case (smallest possible input) + make it do what you want (other steps))
* Ex2: printAscDsc(int start, int end);
  + Solution: cout << start; printAscDsc(start + 1, end); cout << start; (defining the hypothesis for the smaller input - print ascdsc should print start upto end for smaller size input given)
* Ex3: factorial(n);
  + Solution: return n \* factorial (n - 1);
* Ex4: areAllEven(int arr[], int n);
  + Solution: res = areAllEven(arr[], n -1); if (res) { return a[n-1] % 2 ==0 } else false;