ASSIGNMENT NO 3

A) USE OF DIVIDE AND CONQUER STRATEGIES TO EXPLOIT DISTRIBUTED/ PARALLEL/ CONCURRENT PROCESSING OF THE ABOVE

Divide and conquer strategy:

In computer science, **divide and conquer** is an algorithm design paradigm based on multi-branched recursion. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

This divide and conquer technique is the basis of efficient algorithms for all kinds of problems, such as sorting (e.g., quicksort,merge sort),multiplying large numbers(e.g. the Karatsuba algorithm), finding the closest pair of points,syntactic analysis (e.g., top-down parsers), and computing the discrete Fourier transform(FFTs)

Understanding and designing divide and conquer algorithms is a complex skill that requires a good understanding of the nature of the underlying problem to be solved. As when proving a theorem by induction, it is often necessary to replace the original problem with a more general or complicated problem in order to initialize the recursion, and there is no systematic method for finding the proper generalization. These divide and conquer complications are seen when optimizing the calculation of a Fibonacci number with efficient double recursion.

The correctness of a divide and conquer algorithm is usually proved by mathematical induction, and its computational cost is often determined by solving recurrence relations.

Advantages of Divide and conquer strategies:

- **Solving Difficult problems:**

  Divide and conquer is a powerful tool for solving conceptually difficult problems: all it requires is a way of breaking the problem into sub-problems, of solving the trivial cases and of combining sub-problems to the original problem. Similarly, divide and conquer only requires reducing the problem to a single smaller problem, such as the classic Tower of Hanoi puzzle, which reduces moving a tower of height n to moving a tower of height $n-1$.

- **Algorithm efficiency:**

  The divide-and-conquer paradigm often helps in the discovery of efficient algorithms. It was the key, for example, to Karatsuba's fast multiplication method, the quicksort and merge sort algorithms, the Strassen algorithm for matrix multiplication, and fast Fourier transforms.

In all these examples, the D&C approach led to an improvement in the asymptotic cost of the solution. For example, if (a) the base cases have constant-bounded size, the work of splitting the problem and combining the partial solutions is proportional to the problem's size n, and (b) there is a bounded number p of subproblems of size ~ n/p at each stage, then the cost of the divide-and-conquer algorithm will be $O(n \log pn)$.

- **Parallelism:**

  Divide and conquer algorithms are naturally adapted for execution in multi-processor machines, especially shared-memory systems where the communication of data between processors does not need to be planned in advance, because distinct sub-problems can be executed on different processors.

- **Memory access:**

  Divide-and-conquer algorithms naturally tend to make efficient use of memory caches. The reason is that once a sub-problem is small enough, it and all its sub-problems can, in principle, be solved within the cache, without accessing the slower main memory. An algorithm designed to exploit the cache in this way is called cache-oblivious, because it does not contain the cache size as an explicit parameter. Moreover, D&C algorithms can be designed for important algorithms (e.g., sorting, FFTs, and matrix multiplication) to be *optimal*cache-oblivious algorithms–they use the cache in a probably optimal way, in an asymptotic sense, regardless of the cache size. In contrast, the traditional approach to exploiting the cache is *blocking*, as in loop nest optimization, where the problem is explicitly divided into chunks of the appropriate size—this can also use the cache optimally, but only when the algorithm is tuned for the specific cache size(s) of a particular machine.

B) **IDENTIFY THE OBJECTS.**

Objects:

An object can be a variable, a data structure, or a function. In the class-based object oriented programming paradigm,”object” refers to a particular instance of a class where the object can be a combination of a variables, functions, and data structures.

In our project the objects are:

- Python plugins like keras,pybrain
- Firebase
- Labelled health data

C) **IDENTIFY MORPHISM**

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Here we are dealing with various parameters of health given by various users. Every user has differnt risks for each disease. Our system co-relates all risks for one user.

D) **IDENTIFY OVERLOADING**

Overloading:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

In our project overloading is in the python application. The same method is called for different types of disease.

E) **IDENTIFY FUNCTIONAL DEPENDENCIES**

Functional dependencies:

Functional dependency is a relationship that exists when one attribute uniquely determines another attribute. If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X -> Y,which specifies Y is functionally dependent on X.

Our project has functional dependencies of health data for various patients & their risks.