# Activity Scheduling Tool Project

# Algorithm Time Complexity Analysis

# and Correctness

**Project ID: CS311-G30**

2018-CS-22 2018-CS-33

Ayesha Tahir Sapna Urooj Iqbal

Section: A

Subject: Analysis of algorithm

Computer Science and Engineering Department

University of Engineering and Technology, Lahore

# Time complexity Analysis

**Algorithm :**

//function to return teacher id

**return\_tID( labSubj) Cost**

1 Let tid is the teacher id number Ѳ(1)

2 for m = 0 to labSubj.size Ѳ (n)

3 if labSubj[m] == '/' Ѳ(n-1)

4 tid = labSubj.substr(0,m) Ѳ(n-1)

5 return tid Ѳ(1)

T(n)= Ѳ(1)+ Ѳ(n)+ Ѳ(n-1)+ Ѳ(n-1)+ Ѳ(1)

As we ignore the smaller terms so,

**T(n)= Ѳ(n)**

//function to return room number for lab subject

**return\_roomNo( labSubj) Cost**

1 let room is the room number to be allocates Ѳ(1)

2 for m = 0 to labSubj.size Ѳ(n)

3 if labSubj[m] == 'r' Ѳ(n-1)

4 room = stringtointeger(labSubj.substr(m+1)) Ѳ(n-1)

5 return room Ѳ(1)

T(n)= Ѳ(1)+ Ѳ(n)+ Ѳ(n-1)+ Ѳ(n-1)+ Ѳ(1)

As we ignore the smaller terms so,

**T(n)= Ѳ(n)**

//function to generate random integer

**randomint( lower, upper) Cost**

1 srand(time(0)+randomoffset) Ѳ(1)

2 randomoffset = (randomoffset+1)%2823401239LL Ѳ(1)

3 if upper<lower Ѳ(1)

4 return lower Ѳ(1)

5 return rand()%(upper-lower+1)+lower Ѳ(1)

T(n)= Ѳ(1)+ Ѳ(1)+ Ѳ(1)+ Ѳ(1)+ Ѳ(1)

**T(n)= Ѳ(1)**

//function for checking constraints

**randombool(chance) Cost**

1 If randomint(0,1000000) < (1000000\*chance) Ѳ(1)

2 return true Ѳ(1)

3 else Ѳ(1)

4 return false Ѳ(1)

T(n)= Ѳ(1)+ Ѳ(1)+ Ѳ(1)+ Ѳ(1)

**T(n)= Ѳ(1)**

//function to get minimum fitness id

**getminfitnessid() Cost**

1 let minvalue = POSITIVE\_INFINITY Ѳ(1)

2 let minid = 0, count = 0 Ѳ(1)

3 let kteacher, lteacher, ktid, ltid are strings Ѳ(1)

4 let n = 0 Ѳ(1)

5 let kroom, lroom Ѳ(1)

6 let tempfitness = 0, first2Hours = 0, confAvail = 0, oneLabperday = 0 Ѳ(1)

7 for i = 0 to population.size Ѳ(n)

8 tempfitness = 0 Ѳ(n)

9 first2Hours = 0 Ѳ(n)

10 confAvail = 0 Ѳ(n)

11 oneLabperday = 0 Ѳ(n)

12 for j = 0 to labslots Ѳ(n2)

13 for k = 0 to nLabs Ѳ(n3)

14 if population[i].table[k][j] != EMPTY Ѳ(n3)

15 room = return\_roomNo(labTeachers[population[i].table[k][j]]) Ѳ(n3)

16 if initial[room-1][2\*j] != EMPTY || initial[room-1][2\*j+1] !=EMPTY Ѳ(n3)

17 confAvail++ Ѳ(n3)

18 let count = 0 Ѳ(1)

19 for j = 0 to labslots Ѳ(n)

20 If j%(labslots/5) == 0 Ѳ(n)

21 count += 1 Ѳ(n)

22 for k = 0 to nLabs Ѳ(n2)

23 if population[i].table[k][j] == EMPTY Ѳ(n2)

24 continue Ѳ(n2)

25 else Ѳ(n2)

26 kteacher = labTeachers[population[i].table[k][j]] Ѳ(n2)

27 kroom = return\_roomNo(kteacher) Ѳ(n2)

28 ktid = return\_tID(kteacher) Ѳ(n2)

29 for n = j+1 to count\*(labslots/5) Ѳ(n3)

30 for l = 0 to nLabs Ѳ(n4)

31 if population[i].table[l][n] == EMPTY Ѳ(n4)

32 continue Ѳ(n4)

33 else Ѳ(n4)

34 lteacher = labTeachers[population[i].table[l][n]] Ѳ(n4)

35 lroom = return\_roomNo(lteacher) Ѳ(n4)

36 lid = return\_tID(lteacher) Ѳ(n4)

37 if kroom == lroom Ѳ(n4)

38 oneLabperday += 1 Ѳ(n4)

39 If ktid.compare(ltid) == 0 Ѳ(n4)

40 oneLabperday += 1 Ѳ(n4)

41 for j = 0 to labslots Ѳ(n)

42 for k = 0 to nLabs Ѳ(n2)

43 for l = k+1 to nLabs Ѳ(n3)

44 if conflicts[population[i].table[k][j]][population[i].table[l][j]] != 0 Ѳ(n3)

45 confAvail += 1 Ѳ(n3)

46 let firstPeriod, secondPeriod Ѳ(1)

47 for m = 0 to nLabs Ѳ(n)

48 for n = 0 to 5 Ѳ(n2)

49 firstPeriod = n\*labslots/5 Ѳ(n2)

50 secondPeriod = n\*labslots/5+1 Ѳ(n2)

51 if population[i].table[m][firstPeriod] == EMPTY Ѳ(n2)

52 first2Hours += 1 Ѳ(n2)

53 if population[i].table[m][secondPeriod] == EMPTY Ѳ(n2)

54 first2Hours += 1 Ѳ(n2)

55 tempfitness = 0.8\*confAvail + 0.05\*first2Hours + 0.15\*oneLabperday Ѳ(1)

56 population[i].fitness = tempfitness Ѳ(1)

57 if tempfitness < minvalue Ѳ(1)

58 minvalue = tempfitness Ѳ(1)

59 minid = i Ѳ(1)

60 return minid Ѳ(1)

By ignoring the smaller terms, we will get

**T(n)= Ѳ(n4)**

**tournamentselection() Cost**

1 let tournamentminfitness = POSITIVE\_INFINITY Ѳ(1)

2 Let tournamentwinnerid = 0 Ѳ(1)

3 Let tempint a temporary number Ѳ(1)

4 for i = 0 upto tournamentsize Ѳ(n)

5 tempint = randomint(0,population.size()-1) Ѳ(n)

6 if population[tempint].fitness < tournamentminfitness Ѳ(n)

7 tournamentminfitness = population[tempint].fitness Ѳ(n)

8 tournamentwinnerid = tempint Ѳ(n)

9 return tournamentwinnerid Ѳ(1)

T(n)= Ѳ(1)+ Ѳ(1)+ Ѳ(1)+ Ѳ(n)+ Ѳ(n)+ Ѳ(n)+ Ѳ(n)+ Ѳ(n)+ Ѳ(1)

By ignoring the lower terms, we will get

**T(n)=(n)**

//Function for individual crossing over

**crossover(a, b) Cost**

1 let offspring is an individual Ѳ(1)

2 for i = 0 upto nLabs Ѳ(n)

3 let weekperiod is a integer vector Ѳ(n)

4 for j = 0 upto labslots Ѳ(n2)

5 if labInitial[i][j] == EMPTY Ѳ(n2)

6 weekperiod.push\_back(population[b].table[i][j]) Ѳ(n2)

7 for j = 0 upto labslots Ѳ(n2)

8 if labInitial[i][j] != EMPTY Ѳ(n2)

9 offspring.table[i][j] = initial[i][j] Ѳ(n2)

10 else Ѳ(n2)

11 if j < labCrossverSplit Ѳ(n2)

12 offspring.table[i][j] = population[a].table[i][j] Ѳ(n2)

13 erase. (weekperiod.begin(),weekperiod.end(),offspring.table[i][j])) Ѳ(n2)

14 else Ѳ(n2)

15 offspring.table[i][j] = weekperiod[0] Ѳ(n2)

16 weekperiod.erase(weekperiod.begin() Ѳ(n2)

17 return offspring Ѳ(1)

By ignoring the lower terms, we will get

**T(n)= Ѳ(n2)**

**Genetic\_Algorithm() Cost**

1 Let elapsedgenerations = 0 Ѳ(1)

2 Let elitismoffset = 0 Ѳ(1)

3 if(elitism) Ѳ(1)

4 elitismoffset = 1 Ѳ(1)

5 while(elapsedgenerations < generationlimit) Ѳ(n)

6 let vector <individual> newpopulation Ѳ(n)

//compute fitness, find minimum

7 Let minid = getminfitnessid() Ѳ(n5)

8 Let minvalue = population[minid].fitness Ѳ(n)

9 if(elitism) Ѳ(n)

10 newpopulation.push\_back(population[minid]) Ѳ(n)

//crossover

11 For i = elitismoffset upto population.size Ѳ(n2)

12 let a = tournamentselection() Ѳ(n3)

13 let b = tournamentselection() Ѳ(n3)

14 let individual offspring = crossover(a,b) Ѳ(n4)

15 newpopulation.push\_back(offspring) Ѳ(n2)

//mutate

16 For i = elitismoffset upto population.size Ѳ(n2)

17 for( j = 0 to nLabs ) Ѳ(n3)

18 if(randombool(mutationrate)) Ѳ(n3)

19 let a, b Ѳ(n3)

20 do

{

21 a = randomint(0,labslots-1) Ѳ(n4)

22 b = randomint(0,labslots-1) Ѳ(n4)

23 } while((initial[j][a]!=EMPTY) || (initial[j][b]!=EMPTY)) Ѳ(n4)

24

25 swap(newpopulation[i].table[j][a],newpopulation[i].table[j][b]) Ѳ(n2)

26 population = newpopulation Ѳ(n2)

27 elapsedgenerations++ Ѳ(n2)

28 minid = getminfitnessid() Ѳ(n4)

By ignoring the lower terms, we will get

**T(n)= Ѳ(n5)**

As **Genetic\_Algorithm()** is our main function and five phases are considered in a genetic algorithm.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

Moreover, fitness is the prime function and it decides which will be selected and it takes maximum time, which is crucial for the whole algorithm and our algorithm it takes **Ѳ(n5).** A genetic algorithm is a polynomial time algorithm. We are trying to improve the time complexity of our algorithm and we will work to optimize our algorithm to reduce its time complexity.

**Correctness of algorithm**

**Introduction:**

The algorithm takes teachers list, classrooms, subjects, department name, number of lectures in a day etc. as input. The method followed is that in the start of algorithm, a large population of random chromosomes is created. After decoding each will give different solution to the problem.

In algorithm, first of all, an initial generation of chromosomes is created randomly and their fitness values are analyzed. After this the new Generations created performs some operations for each generation such as preserving few fittest chromosomes from the previous generation for the new generation. After iterations we may not get optimal solution so, some characteristics are preserved from the previous one in the coming generations. Then a pair is randomly selected from previous generation. It will perform mutation on the more fit chromosome. And then analyze the fitness of the new generation of chromosomes and order them according to fitness values. So, in order to find a final optimal solution, the whole process will repeat until the chromosomes of desired fitness value are found and the result generated is optimal.

**Correctness of algorithm**

1. Inductive Hypothesis:

After adding desired values in the required files, the algorithm will generate a timetable that will deal with classes occupied, teachers credit hours etc.

1. Base case:

If zero activities are added to algorithm it will generate an optimal solution.

1. Inductive step:

If we generate a k timetable with input values that is the best solution, then the next one k+1 generated will also be the best one.

1. Conclusion:

The computational effort of algorithm which is related to the initial population and number of iterations can also deal with the performance or accuracy of algorithm.

The constraints, range and number of inputs as well as the structure of optimization algorithm are related to the performance and we may do changes in algorithm to make it work more efficiently. And after all the activities are entered, the algorithm will provide the optimal or best solution.

We never rule the solution. We got some solutions at the end of algorithm and we expect them to be the best solution to our problem. We may make some changes in algorithm to give the best solution.