Analysis the method day() and related methods
Create by Tan Pham
01/28/2015

```
***************************************************************************************
    public void day(){

        this.sort(); //According to calculating below, this function has O(n^2) runtime--------> O(n^2)

        mostFit = population.get(0);                                          ---------> constant1

        population.subList(population.size()/2, population.size()).clear();          ---------> linear1

        //This loop goes n/2 times
        //According to calculating below,
        //the mutate() function has a constant runtime.
        //the crossover(other) has a constant runtime.
        while(population.size() < popSize){                                   ----------> constant2
          if(rand.nextBoolean()){                                             ----------> constant3
            Genome newGen = new Genome(population.get(rand.nextInt(population.size()))); -->constant4
            newGen.mutate();                                                  ---------> constant5
            population.add(newGen); //                                          -----> amotized const
          }else{
            Genome newGen1 = new Genome(population.get(rand.nextInt(population.size()))); -->constant6
            Genome newGen2 = new Genome(population.get(rand.nextInt(population.size()))); -->constant7
            newGen2.crossover(newGen1);                                           ---->constant8
            newGen2.mutate();                                                  ----------->constant9
            population.add(newGen2);                                             -----> amotized const
            }
        }
    }

=========>>> Total time = constant1 + linear1 + n/2*sum(constant2 through constant9, amotized const) + O(n^2)
            = constant1 + linear1 + linear2 + O(n^2)
            = O(n^2)


***************************************************************************************
    /**
     * Sort the population in increasing fitness order.
     */
    public void sort(){

        //This loop goes n times.
        //According to calculating below,
        //the swap() function has a constant runtime.
        //the findMin(startIndex) has a linear runtime
        for(int i = 0; i< population.size(); i++) {                  //constant1
            int mindex = findMin(i);                               //linear
            if (mindex != i){                                    //constant3
                swap(population.get(i), population.get(mindex));//constant4
            }
        }
    }

=============>> Total time = n*sum(constant1,constant3,constant4, linear)
                    = n*(constant + O(n))
                    = n*O(n)
                    = O(n^2)



****************************************************************************************************
    /**
     * Find the min fitness in the population start from startIndex.
     * @param startIndex
     * @return the index of the min fitness element.
     */
    private int findMin(int startIndex) {
        int minFit = population.get(startIndex).fitness(target);       //constant1
        int mindex = startIndex;                                       //constant2
        int i;                                                 //constant3

        //this loop goes n time for the worst case.
        //According to calculating below,
        //the fitness(target) function has a constant runtime.
        for(i = startIndex; i < population.size(); i++) {           //constant4
            if(minFit > population.get(i).fitness(target)) {       //constant5
                minFit = population.get(i).fitness(target);     //constant6
                mindex = i;                                    //constant7
            }
        }
        return mindex;                                               //constant8
    }

===========>> Total time = sum(constant1,constant2,constant3,constant8)
                    + n*sum(constant4,constant5,constant6,constant7)
                   = constant9 + n*constant10
                   = O(n)    -----> linear



****************************************************************************************************
    /**
     * swap the genome g1 and g2 in the population.
     * @param g1
     * @param g2
     */
    private void swap(Genome g1, Genome g2){
        String temp;                        //constant
        temp = g1.genome;                       //constant
        g1.genome = g2.genome;                     //constant
        g2.genome = temp;                       //constant
    }

=========>> Total time: constant
```

```
*************************************************************************************************
    /**
     * Calculate the fitness of the current string to the target.
     * @param target
     * @return
     */
    public int fitness(String target){                                  //constant1
        int fitness = Math.abs(genome.length() - target.length() );     //constant2
        int length;                                                     //constant3
        if (genome.length()>target.length()) {                          //constant4
            length = genome.length();                                   //constant5
        }else{
            length = target.length();                                   //constant6
        }

        //Since the name have a limited constant characters,
        //so this loop goes a constant13 times.
        for(int i = 0; i < length;i++){                                 //constant7
            if (i >= genome.length()||i >= target.length()){            //constant8
                fitness += 1;                                           //constant9
            }else if(genome.charAt(i) != target.charAt(i)){             //constant10
                fitness += 1;                                           //constant11
            }
        }
        return fitness;                                                 //constant12
    }

================>>> Total time = sum(constant1 through constant6, constant12)
                    + constant13*sum(constant7 through constant11)
                   = constant



*************************************************************************************************
    /**
     * Mutation method.
     */
    public void mutate(){


        if(rand.nextDouble()<=mutationRate){                                    //constant
            genome = randAdd(genome);                                           //constant
        }


        if(rand.nextDouble()<=mutationRate){                                    //constant
            if(genome.length()>2) genome = randDel(genome);                          //constant
        }


        if(rand.nextDouble()<=mutationRate){                                    //constant
```

```
            if(genome.length()>1) genome = randRep(genome);            //constant
        }
    }

================================>> Total time = constant


*******************************************************************************************
    /**
     * Crossover method.
     * @param other
     */
    public void crossover(Genome other){
        String newStr = "";                                        //constant
        int index = 0;                                             //constant

        //Since the name have a limited constant characters,
        //so this loop goes a constant times.
        while(index < genome.length()){                                 //constant
            if(rand.nextBoolean()){                                  //constant
                newStr += genome.charAt(index);                         //constant
            }else{
                if(index < other.genome.length()){                      //constant
                    newStr += other.genome.charAt(index);                 //constant
                }else break;                                       //constant
            }
            index++;                                               //constant
        }
        genome = newStr;                                           //constant
    }

==============================>> Total time = constant


*******************************************************************************************
    private Character randGetChar(){
        return list[rand.nextInt(28)];
    }

========================>> Total time = constant


*******************************************************************************************
    /**
     * Add a random selected character to a random position of a string.
     * @param str - original string
     * @return new string
     */
    private String randAdd(String str){                                //constant
        String newStr, tempStr;                                    //constant
        tempStr = str.substring(0, rand.nextInt(str.length()));         //constant
        str = str.substring(tempStr.length());                     //constant
        newStr = tempStr + randGetChar() + str;                      //constant
        return newStr;                                          //constant
        }

=========================>> Total time = constant


*******************************************************************************************
        /**
         * Delete a random selected character from a random position of a string.
         * @param str - original string
         * @return new string
         */
        private String randDel(String str){                                    //constant
                String newStr, tempStr;                                        //constant
                tempStr = str.substring(0, rand.nextInt(str.length()));               //constant
                str = str.substring(tempStr.length()+1);                           //constant
                newStr = tempStr + str;                                        //constant
                return newStr;                                             //constant
        }

==============================> Total time = constant


*******************************************************************************************
        /**
         * Replace a random selected character at a random position of a string.
         * @param str - original string
         * @return new string
         */
        private String randRep(String str){                                    //constant
                String newStr, tempStr;                                        //constant
                tempStr = str.substring(0, rand.nextInt(str.length()));               //constant
                str = str.substring(tempStr.length()+1);                           //constant
                newStr = tempStr + randGetChar() + str;                             //constant
                return newStr;                                             //constant
        }
}


=========================>> Total time = constant
```