# Design and Analysis of Algorithm

## Subset Sum Problem

Duong Van Thang - Vo Quang Vinh

December 18, 2019

# Table of Contents

# Table of Contents

# Subset Sum Problem

## Description

Given a set of N positive integers and a positive integer S. Is there any non-empty subset whose sum is the same as S ?

## Why?

- The problem is NP-complete.
- Subset sum can be thought of as a special case of the knapsack problem.
- Special cases of subset sum are the partition problem.

# Subset Sum Problem

## Input

- The first line contain to two positive integers N and S.
- The second line contain N positive integers.

## Output

Print "Yes" if there exists a subset whose sum is equal to S. Otherwise, print "No".

# Table of Contents

# Solving Problem

- Math Model
- Backtracking Algorithm
- Dynamic Programming Algorithm
- Greedy Algorithm
- Genetic Algorithm

# Mathematics

## Math Model

- Problem instance:
  + W = $\{w_1, w_2, ..., w_n\}$ where $w_i$'s are positive integers.
  + S is a large positive.
- Feasible solution:
  $\overrightarrow{X} = \{x_1, x_2, ..., x_n\}$ where $x_i \in \{0; 1\}$ and satisfied the condition.
- Condition:

$$\sum_{i=1}^{n} w_i * x_i \leq S$$

- Optimal solution:

$$\sum_{i=1}^{n} w_i * x_i \longrightarrow MAX$$

# Backtracking

## Method

- Generate all the binary strings of N bits.
- sum(i) $= \sum_{j=0}^{i} x_j * w_j$.
- exists(): print "Yes" and save the result.
- When sum(i) = S: call exists() and stop the program.
- Complexity $O(2^n)$.

```
void Try(int i){
    for(int j = 0; j <= 1; j++){
        x[i] = j;
        if(i == n-1){
            if(sum(i) == S){
                exists();
                return;
            }
        }
        else if(sum(i) == S) exists();
        else if(sum(i) < S) Try(i + 1);
    }
}
```

Listing 1: Code C++

# Dynamic Programming

## Method

- Boolen array:
  L(i,j): exists or not a subset of W[1..i] have sum = j.
- Formula Retrieval:

$$\text{L(i,j)} = \begin{cases} TRUE & \text{,if } j = 0 \\ FALSE & \text{,if } i = 0 \\ L(i-1, S) \vee L(i-1, S - W[i]) & \text{, otherwise} \end{cases}$$

- If L(n,S) = TRUE then the answer is "Yes" and otherwise.
- Complexity O(n*S).

# Dynamic Programming

Sum

| Elements | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | T | F | F | F | F | F | F |
| 3 | T | F | F | T | F | F | F |
| 2 | T | F | T | T | F | T | F |
| 7 | T | F | T | T | F | T | F |
| 1 | T | T | T | T | T | T | T |

Figure: Examples

```
L[t]:=0; L[0]:=1;

for i := 1 to n do
    for t := S downto a[i] do
            if (L[t]=0) and (L[t-a[i]]=1) then L[t]:=1;
```

Listing 2: Code C++

# Greedy

## Greedy

- We wish to find a subset of W.1,...,W.N whose sum is as large as possible but not larger than T (capacity of the knapsack).
- Consists in examining the items and inserting each new item into the knapsack if it fits.

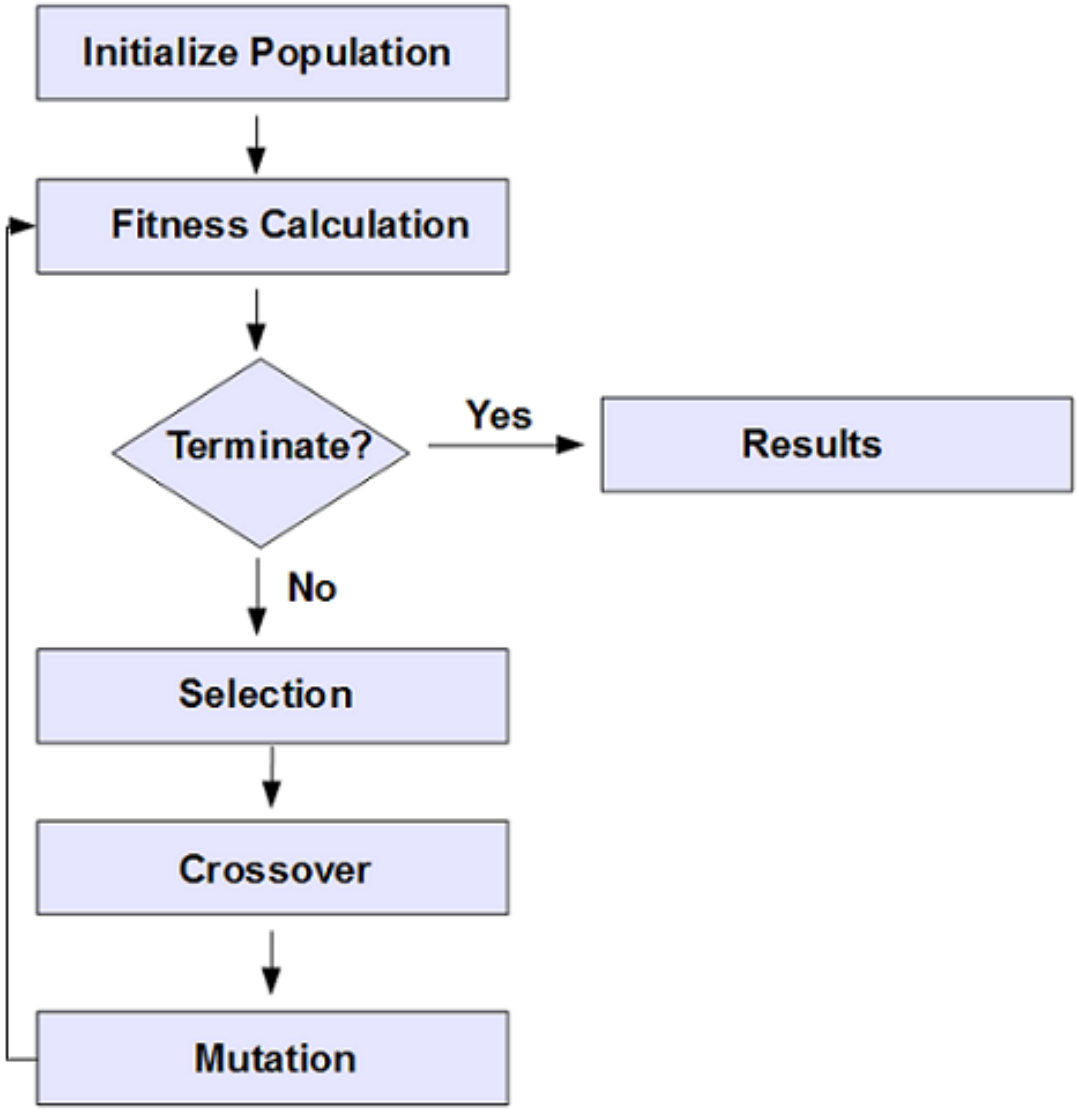
## Method

1.Order $a_j$ by size: $a_1 \geq a_2 \geq \ldots$
2.Introduce s = 0, current sum and j = 1, current index.
3.If s+$a_j$ < b, include $a_j$: that is, s = s+ $a_j$.
4.Increment j.
5.If j < n, then return to step 3, otherwise stop.

```
int Findsubset(int W[], int n, int S)
{
    QUICKSORT(W, 0, n-1);
    int X[n];
    for(int i = 0; i < n; i++)
        X[i] = 1;
    for(int j = 0; j < n; j++){
        if(W[j] > S){
            X[j] = 0;
        }
        else S = S - W[j];
    }
    return  S == 0;
}
```

Listing 3: Code C++

# Genetic Algorithm



# Genetic Algorithm

## Method [1]

- Chromosome: binary string of length n.
- Fitness function:

$$F(\vec{x}) = \begin{cases} S - sum(\vec{x}) & \text{,if } sum(\vec{x}) \leq S \\ sum(\vec{x}) & \text{,otherwise} \end{cases}$$

- Population: 100 individuals.
- Selection: Elitism and Random Selection.
- Crossover: One Point Crossover.
- Mutation: Interchanging.

# Genetic Algorithm

## Improve [1]

- *Setting different-degree*: $D_s = 0.6$.
  + $D_s$ reduce to near zero.
- The *different-degree* ($d_i$) of $i$th parent pair: $d_i = \frac{N_d}{N_g}$ .
  + $N_g$ is the size of chromosome.
  + $N_d$ is the number of different genes between the two parents.
- if $d_i \leq$D$_s$ : mutation, if $d_i > D_s$ : crossover.
- More efficient search and prevents the GAs from falling into local extremes.

Wang, R. L. (2004). *A genetic algorithm for subset sum problem.* Neurocomputing, 57, 463–468.

# Table of Contents

# Analysis

## Analysis

- Backtracking Algorithm.
- Dynamic Programming Algorithm.
- Greedy Algorithm.
- Genetic Algorithm.
- Testing and comparing.

# Backtracking

## Advantages

- Simple installation.
- Always find the correct solution.

## Disadvantages

- The large execution time.
- Requires large amount of space.

# Dynamic Programming

## Advantages

- The small execution time.
- The Pseudo-polynomial Complexity.

## Disadvantages

- Difficult to understand and installation.
- No general formation of Dynamic Program is available.

# Greedy

## Advantages

- The small execution time.
- Finding solution is easy.
- Complexity of Pseudo-polynomial.

## Disadvantages

- Local solution.
- Hard to proving that a greedy algorithm is correct.

# Genetic Algorithm

## Advantages

- Can solve problems with the large parameters.
- Can find fit solutions in a very less time.

## Disadvantages

- Difficult to install.
- Sometime can't guarantee an optimal solution.

# Testing

## Test case 1: "Yes"

5 22
1 2 4 8 16 32

## Test case 2: "Yes"

10 50
25 27 1 12 6 15 9 30 21 19

## Test case 3: "Yes"

10 3368
849 977 686 617 788 465 497 550 262 153

## Test case 4: "Yes"

15 1060
73 703 685 652 696 506 163 503 644 413 38 631 311 112 948

# Testing

## Test case 5: "Yes"

20 5351
510 469 938 280 941 38 938 910 987 613 192 297 681 645 871 319 817 66 599 413

## Test case 6: "No"

20 15891
544 933 538 13 294 928 189 619 479 841 973 69 155 60 9 732 693 439 281 849

## Test case 7: "Yes"

20 55799
2263 6899 7794 8711 1040 2296 3839 1369 5109 1932 3249 9724 2243 9930 3803 4583 8767 7034 4610 3525

# Testing

## Test case 8: "Yes"

20 31629
4515 8029 6146 6606 9899 5628 3271 7837 4885 5773 2314 3510 8847
7291 4411 3405 8516 887 5293 2707

## Test case 9: "No"

20 888496
418276 308806 328906 252155 189153 50630 208338 426296 216014
343742 84214 401001 103832 329783 83232 365090 422036 287385
147092 125945

## Test case 10: "No"

25 1100000
463492 241553 220858 148206 269492 284046 438305 403024 520912
335202 274732 533082 390214 373248 118080 331996 470232 538955
269930 559731 248880 411950 9781 362902 9005

# Testing

## Test case 11: "Yes"

30 3586331
557190 357829 201610 371568 739107 178392 824621 291281 225652
124360 925353 399095 178837 527280 719130 517416 834581 971332
585968 667332 170068 127639 342387 484665 841976 747802 398141
428719 289508 34960

## Running

- Using C++ on a PC station.
- Unit of measure times: MS(millisecond).
- DPA: Dynamic programming.
- GS: Greedy.
- ES: Backtracking.

# Testing


Figure: Running time for test 1,2,3,4,5

# Testing

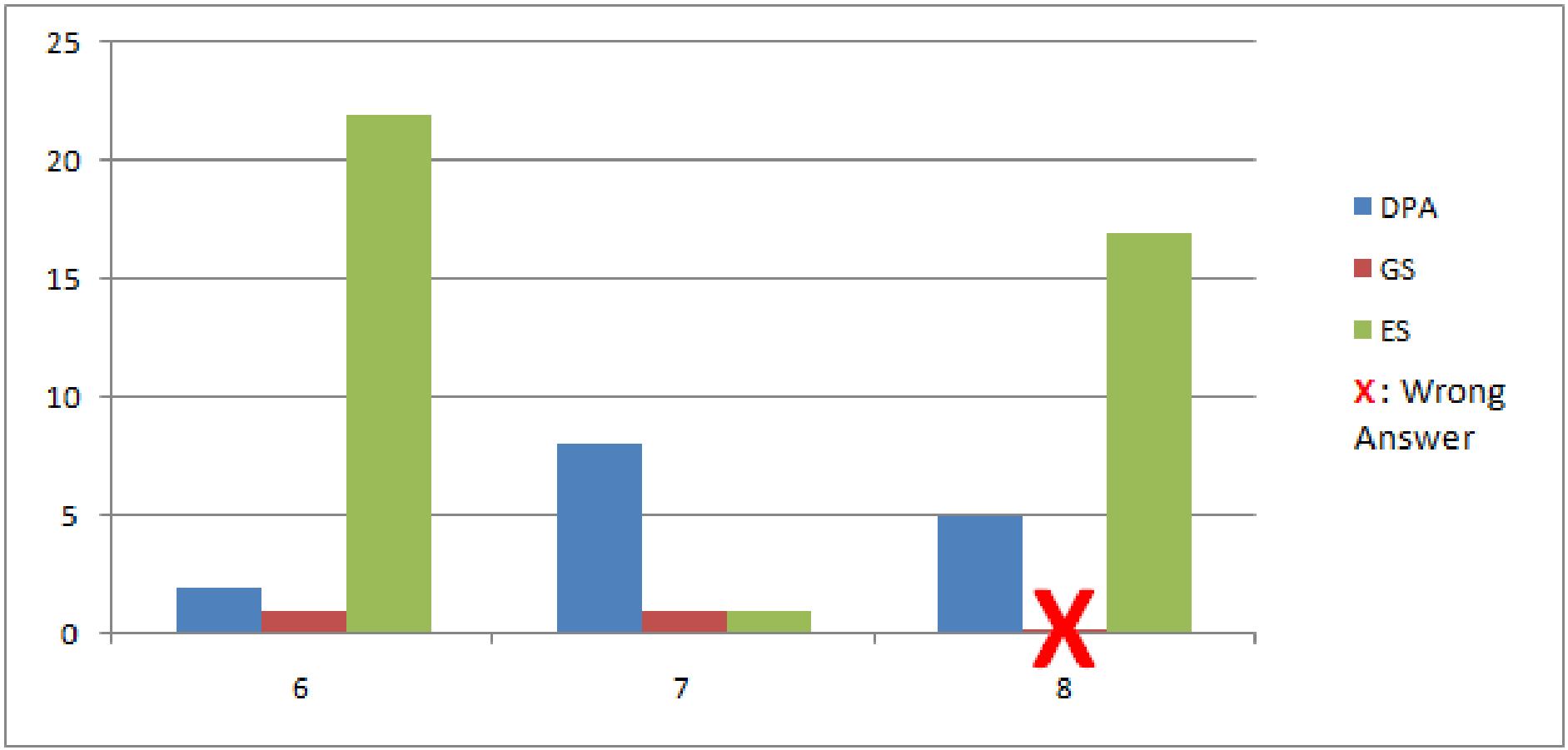



Figure: Running time for test 6,7,8

# Testing

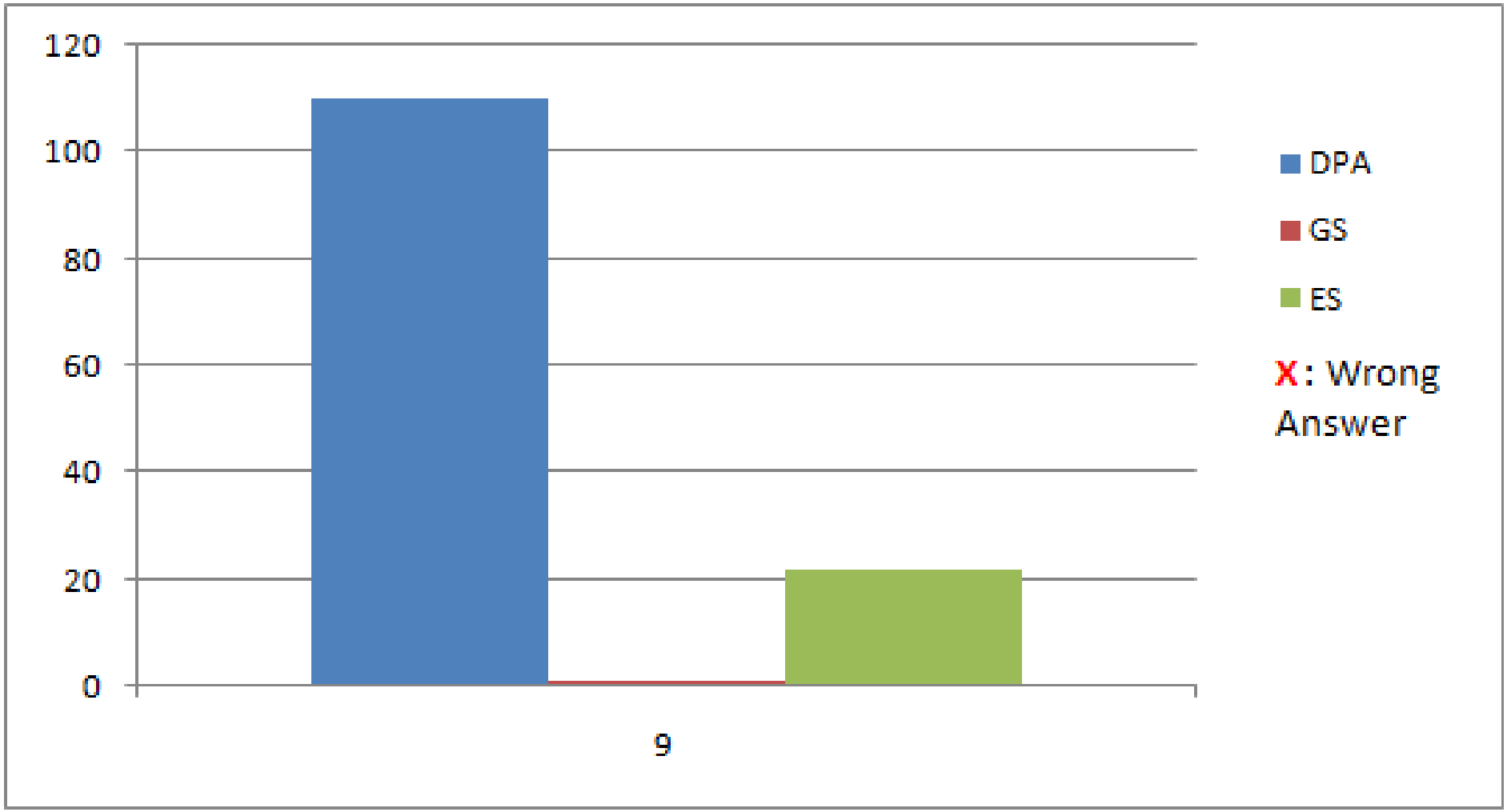



Figure: Running time for test 9

# Testing

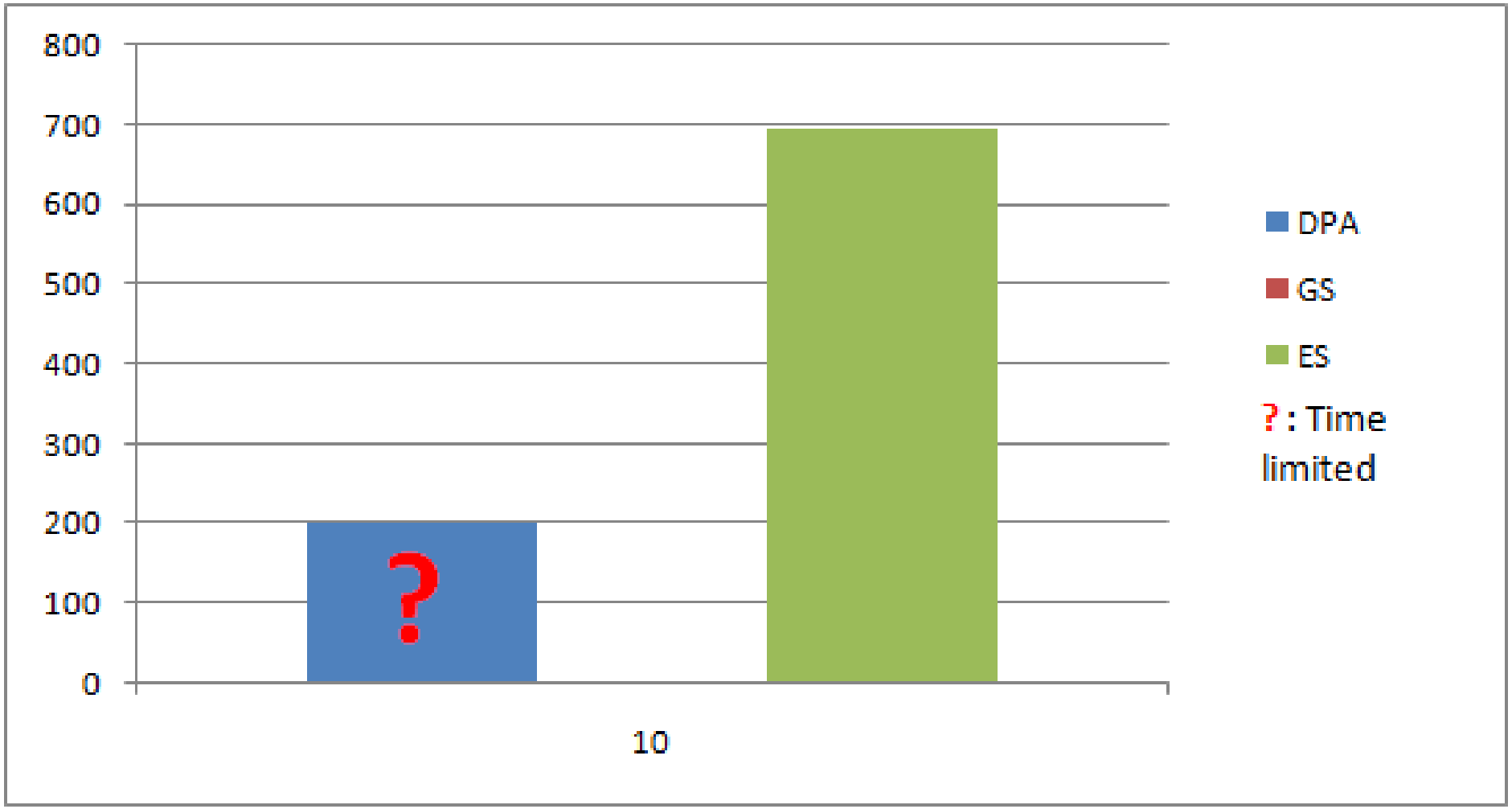



Figure: Running time for test 10

# Testing

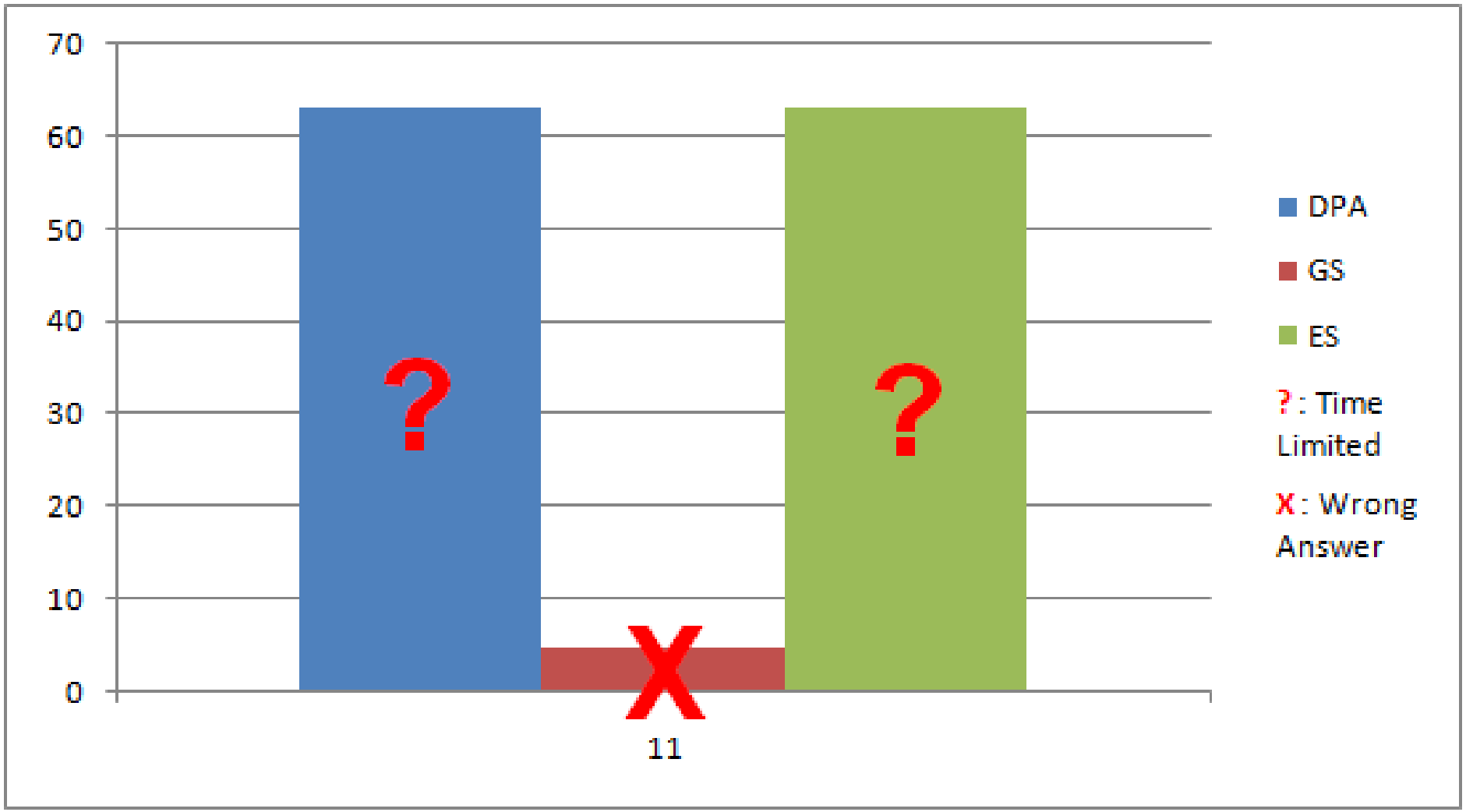



Figure: Running time for test 11

# Testing

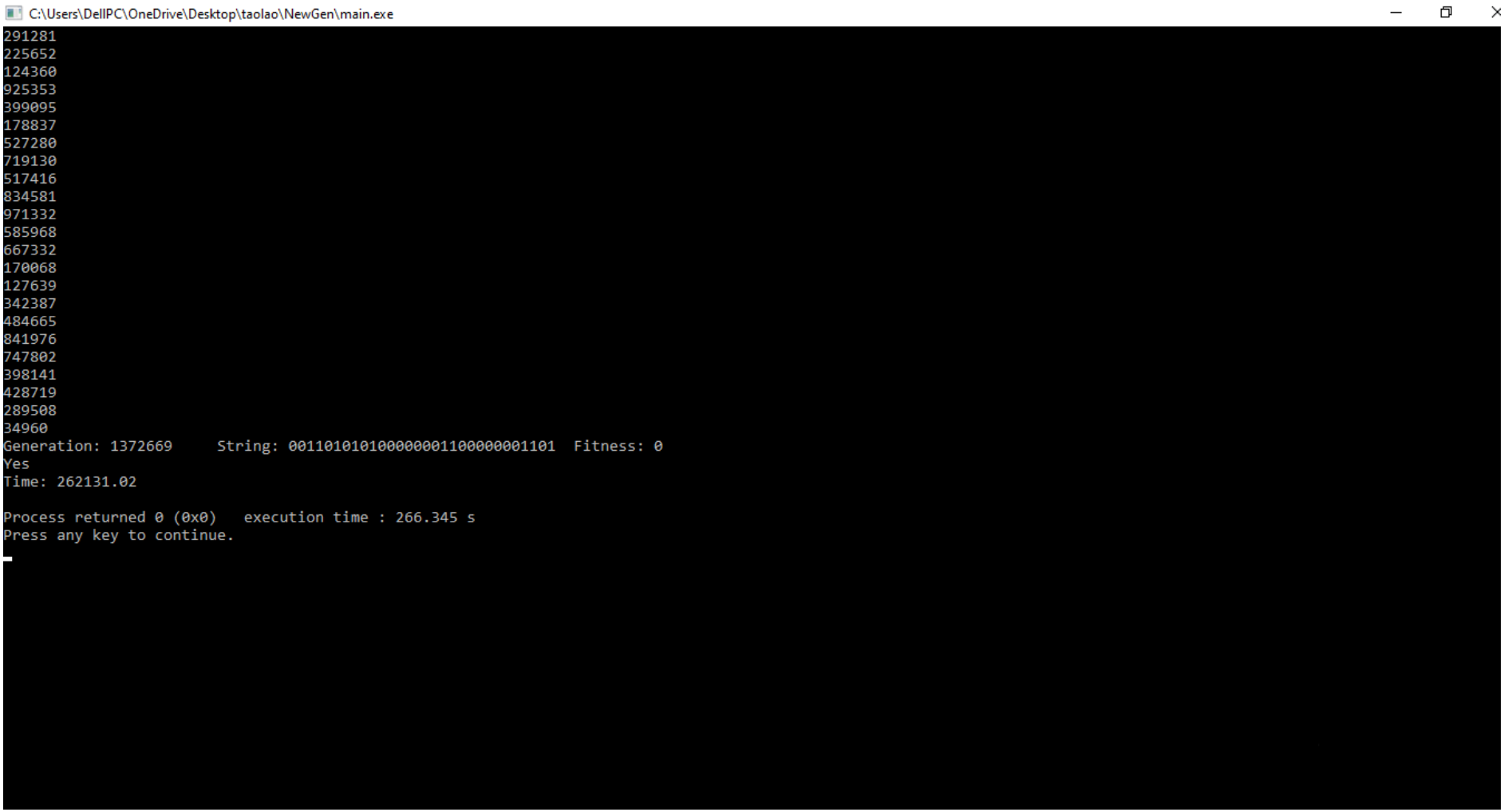



Figure: Genetic Algorithm for test 11

# Table of Contents

# Conclusion

| | Time executed(ms) | Accepted |
|---|---|---|
| Dynamic Programming | 126.02 | 09/11 |
| Backtracking Algorithm | 736.25 | 10/11 |
| Genetic Algorithm | 1168306.02 | 11/11 |
| Greedy Algorithm | 7.20 | 07/11 |

Figure: Conclusion result

# Conclusion

## Best Algorithm

- DPA is the best algorithm in those test cases.

## Conclusion

- Greedy Algorithm can't solve this problem, accuracy about 50%.
- DPA is the best algorithm in limited cases.
- Using Backtracking with small number of elements.
- Using GA for finding solution with the large cases.

# Conclusion

**DEMO**