Assignment 2

Sub-set Sum

Matjaž Mav (63130148) May 23, 2020

1 DYN

This approach has time complexity of $\mathcal{O}(nk)$ and space complexity of $\mathcal{O}(k)$. It is greatly effected by the size of array ***A*** and/or targeted sum ***k***. If we introduce big numbers this problem becomes really hard for this approach. Example of such problem would be:

```
5
10000000
2000000
2000000
2000000
2000000
2000000
```

2 EXH

This approach has time and space complexity of $\mathcal{O}(2^n)$. The worst case scenario is when the input is a long array ***A*** of some relatively small numbers and some really big targeted sum ***k***. If the targeted sum ***k*** is greater or equal then sum of the whole array ***A***, this approach will perform exhaustive search without any kind of pruning. Example of such problem would be:

```
3000000
6000000
1
2
3
1
2
3
...
1
2
3
```

3 GREEDY

This approach has time complexity of $\mathcal{O}(n \log n)$ and space complexity of $\mathcal{O}(n)$. In the worst case this algorithm returns at most 2 times worse result. Example of such problem would be:

```
3
1000000
500001
500000
500000
```

In this case optimal result is 1000000 but the algorithm would return 500001. The result is almost 2 times worse then the optimal ($500001 \approx 1000000/2$).

4 FPTAS

First, we compared how the ϵ parameter effect the execution time of this algorithm on sorted lists. For this evaluation we have generated random inputs using code listed in the Listing 1 and then sort the elements in ascending and descending order. We observed (Figure 1 and Figure 2) that on list sorted in descending order algorithm performs quicker and have much more stable execution time.

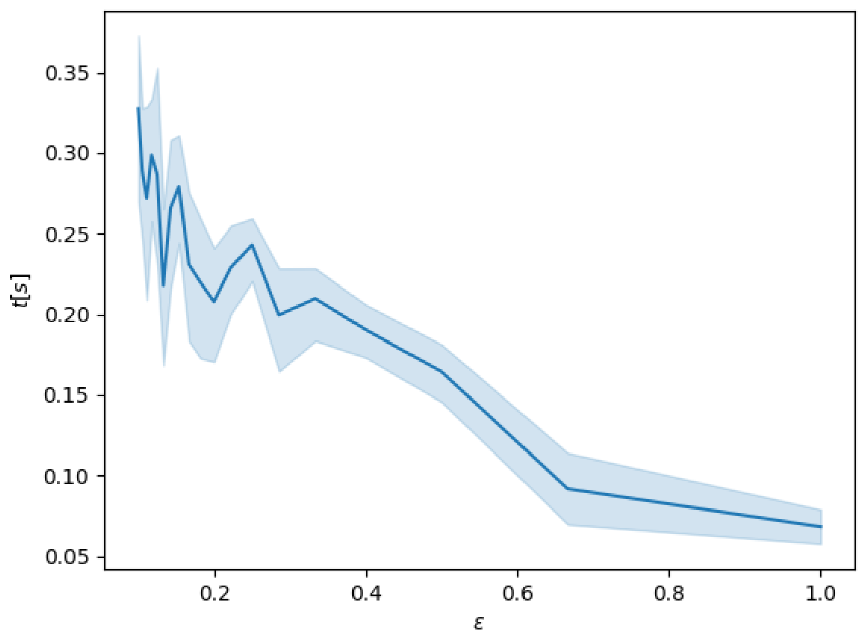

Figure 1: Elements are sorted in ascending order. Execution time compared to the parameter ϵ

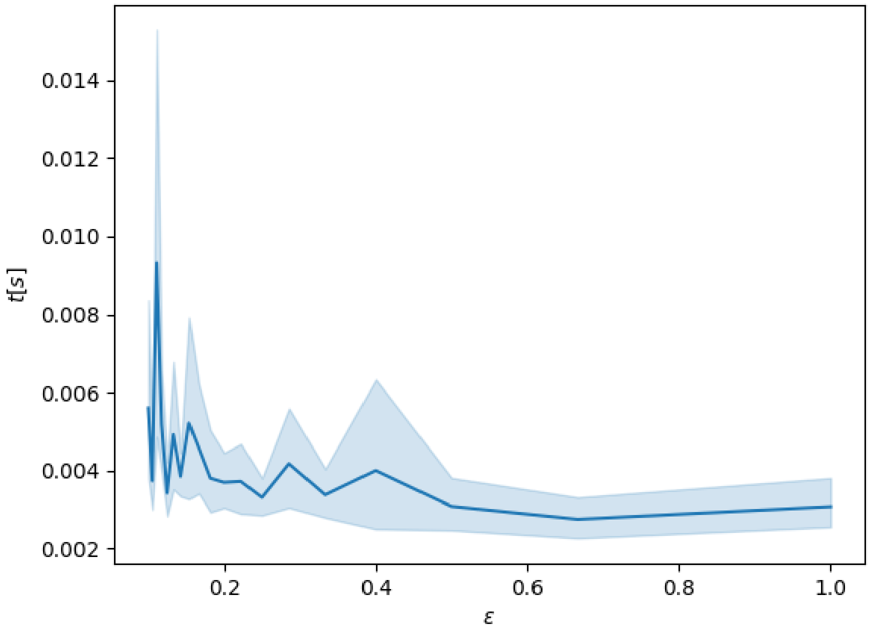

Figure 2: Elements are sorted in descending order. Execution time compared to the parameter ϵ

Second, we upgrade our FPTAS implementation and first sorted elements in descending order. Then we generated exponential inputs using code listed in the Listing 2. From the Figure 3 we can observe that FPTAS algorithm on the exponential generated inputs runs nearly 10 times slower on the random generated inputs.

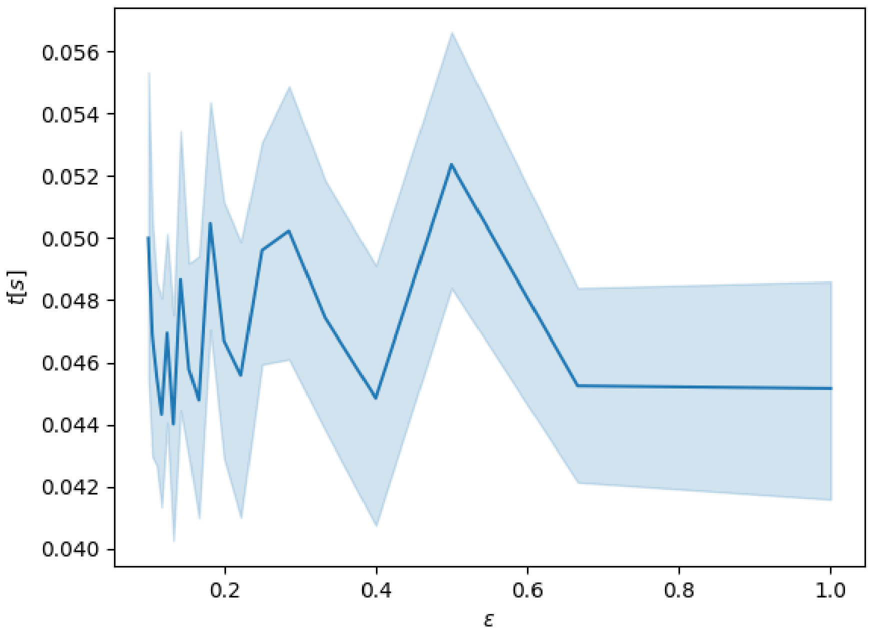

Figure 3: Execution time compared to the parameter ϵ

Listing 1: Random problem generator

```python
def generate(n: int) -> Tuple[List[int], int]:
  A = []
  for _ in range(n):
    A.append(randint(1, int(n/2)))

  k = sum(sample(A, int(n*random())))
  return (A, k)
```

Listing 2: Exponential problem generator

```python
def generate(n: int) -> Tuple[List[int], int]:
  A = list(map(lambda x: x**2, range(n)))
  k = sum(sample(A, int(n*random())))
  return (A, k)
```