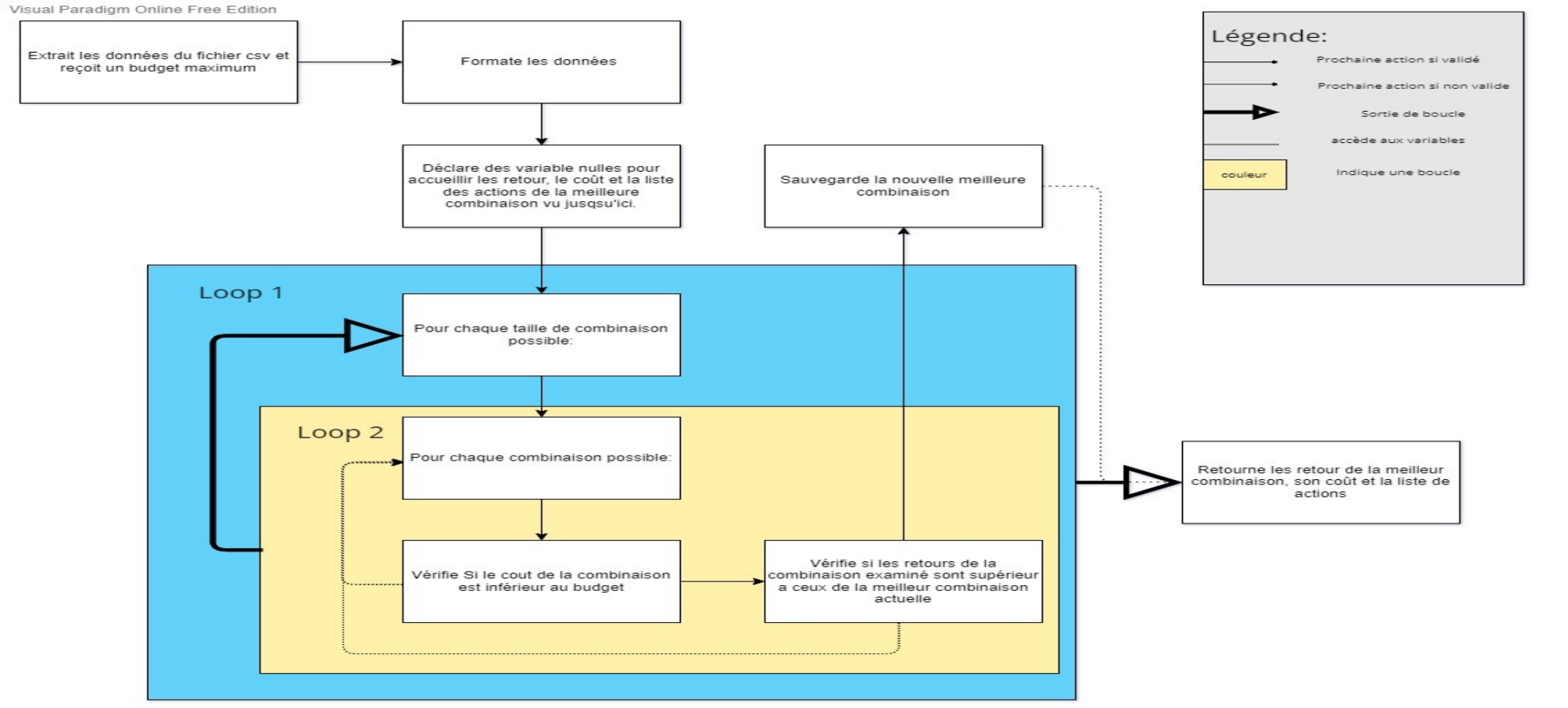
Organigramme : Bruteforce.py



Résultats : Bruteforce.py

```
cout de la combinaison: 498.0
retours de la combinaison: 99.08000000000001
profit de la combinaison: 19.89558232931727
liste des actions:
[{'name': 'action-04', 'price': 70.0, 'profit': 20.0, 'returns': 14.0}, {'name': 'action-05', 'price': 60.0, 'profit': 17.0, 'returns': 10.200000000000001}, {'name': 'action-06', 'price': 80.0, 'profit': 25.0, 'returns': 20.0}, {'name
': 'action-08', 'price': 26.0, 'profit': 11.0, 'returns': 2.86}, {'name': 'action-10', 'price': 34.0, 'profit': 27.0, 'returns': 9.18}, {'name': 'action-11', 'price': 42.0, 'profit': 17.0, 'returns': 7.140000000000001}, {'name': 'acti
on-13', 'price': 38.0, 'profit': 23.0, 'returns': 8.74}, {'name': 'action-18', 'price': 10.0, 'profit': 14.0, 'returns': 1.4000000000000001}, {'name': 'action-19', 'price': 24.0, 'profit': 21.0, 'returns': 5.04}, {'name': 'action-20',
 'price': 114.0, 'profit': 18.0, 'returns': 20.52}]
```

Coût total de la combinaison : 498.00 €
Retours sur la combinaison : 99,08
Profits réalisés : 19,89 %
Liste de actions sélectionnées :
-Action 4,Action 5,Action 6,Action 8,Action 10,Action 11,Action 13,Action 18,Action 19,Action 20.

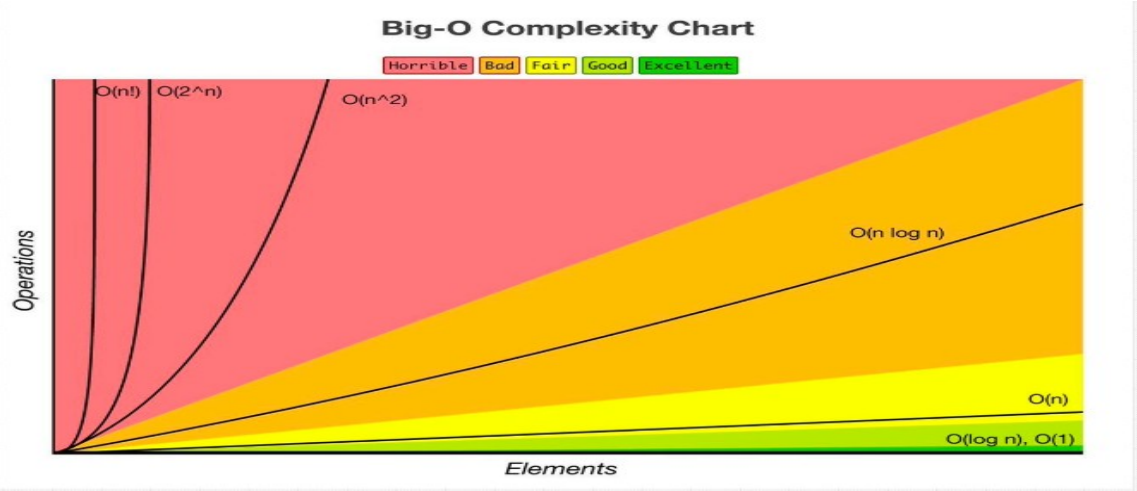
Mesures performance : Bruteforce.py

```
         1862094 function calls in 2.964 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    2.964    2.964 <string>:1(<module>)
   813347    0.842    0.000    0.842    0.000 bruteforce_itertools.py:21(calculate_comb_returns)
  1048576    1.271    0.000    1.271    0.000 bruteforce_itertools.py:31(calculate_comb_cost)
        1    0.000    0.000    0.000    0.000 bruteforce_itertools.py:41(calculate_profit)
        1    0.848    0.848    2.962    2.962 bruteforce_itertools.py:49(bruteforce_itertools)
        1    0.000    0.000    2.964    2.964 bruteforce_itertools.py:80(main_itertools)
        1    0.000    0.000    0.000    0.000 bruteforce_itertools.py:9(prepare_list)
        1    0.000    0.000    0.000    0.000 codecs.py:260(__init__)
        3    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
       21    0.000    0.000    0.000    0.000 csv.py:107(__next__)
        1    0.000    0.000    0.000    0.000 csv.py:81(__init__)
        1    0.000    0.000    0.000    0.000 csv.py:90(__iter__)
       41    0.000    0.000    0.000    0.000 csv.py:93(fieldnames)
        1    0.000    0.000    2.964    2.964 main.py:15(main_set_0_BF_itertools)
        1    0.000    0.000    0.000    0.000 utils.py:4(extract_data)
        3    0.000    0.000    0.000    0.000 {built-in method _codecs.charmap_decode}
        1    0.000    0.000    0.000    0.000 {built-in method _csv.reader}
        1    0.000    0.000    2.964    2.964 {built-in method builtins.exec}
       41    0.000    0.000    0.000    0.000 {built-in method builtins.len}
       22    0.000    0.000    0.000    0.000 {built-in method builtins.next}
        5    0.002    0.000    0.002    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {built-in method io.open}
        1    0.000    0.000    0.000    0.000 {method '__exit__' of '_io._IOBase' objects}
       20    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    15     41.1 MiB     41.1 MiB           1   @profile
    16                                         def main_set_0_BF_itertools():
    17     41.1 MiB      0.1 MiB           1       main_itertools(500, "E:\L7\Livrable\data\dataset0.csv")
```

Durée d'éxécution : 2,964 sec
Utilisation mémoire : 41,1 MiB
(Mesure effectuées avec cProfile & memory_profiler)

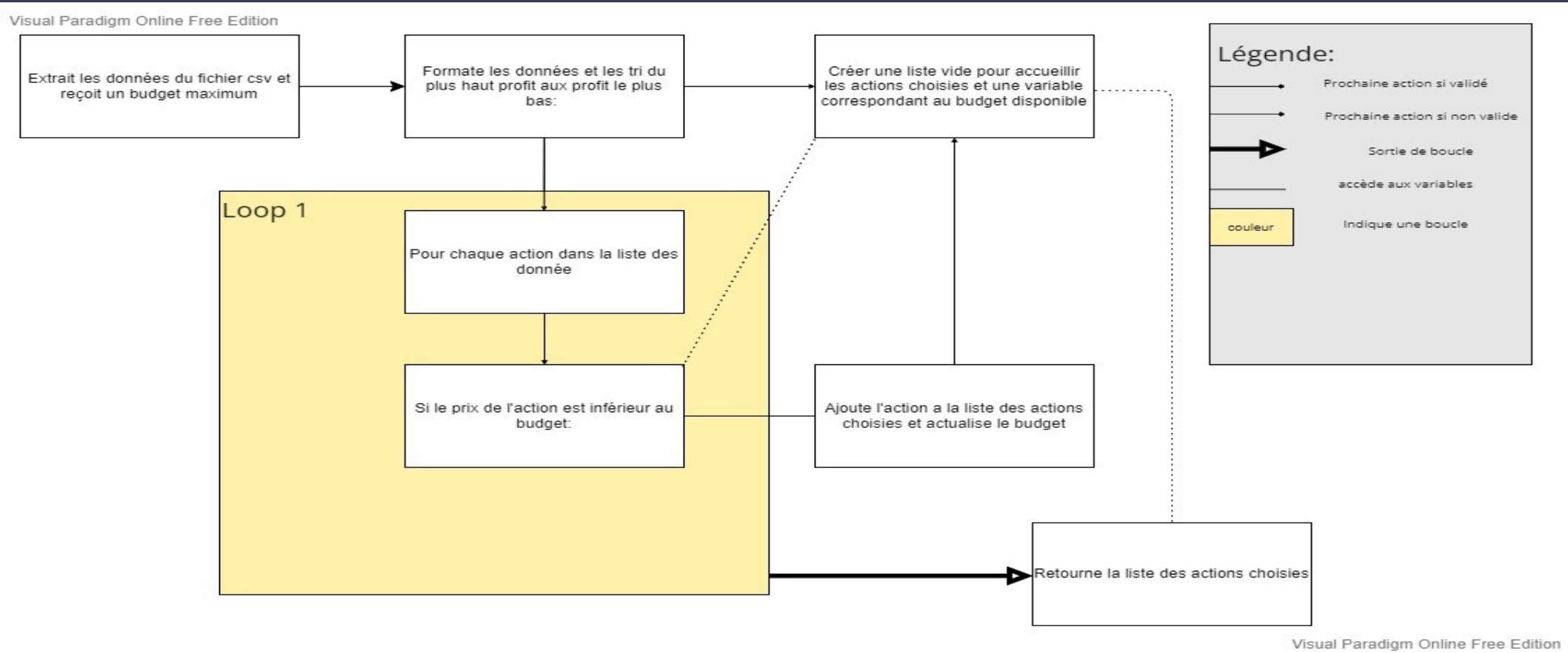
Analyse Big-O : Bruteforce.py

```python
prepared_list = prepare_list(action_list)  # O(n)
combination_count = 0  # O(1)
best_returns = 0  # O(1)
best_comb_cost = 0  # O(1)
comb_with_best_returns = []  # O(1)
for combination_size in range(0, len(prepared_list)+1):  # O(n)
    for combination in itertools.combinations(prepared_list, combination_size):  # O(n!)
        combination_count += 1  # O(1)
        comb_as_list = list(combination)  # O(1)
        comb_cost = calculate_comb_cost(comb_as_list)  # O(1)
        if comb_cost <= budget:  # O(1)
            comb_returns = calculate_comb_returns(comb_as_list)  # O(1)
            if comb_returns > best_returns:  # O(1)
                best_returns = comb_returns  # O(1)
                best_comb_cost = comb_cost  # O(1)
                comb_with_best_returns = comb_as_list  # O(1)
# O(n)+(O(1)*4)+(O(n)*O(n)*O(1)*9 = O(n)+O(1)+(O(n²)*O(1)) = O(n!)
return best_returns, best_comb_cost, comb_with_best_returns
```



Nous gardons l'ordre de grandeur principal:Cet algorithme démontre une complexité de O(n!), Plus l'ensemble de donnée sera grand,plus le temps d'éxécution augmentera de manière exponentielle

Solution optimisé : Greedy : Organigramme



Solution optimisé : Greedy: résultats :

```
---- Data set 1 ----

cout de la combinaison: 499.9400000000001
retours de la combinaison: 198.507805
profit de la combinaison: 39.70632575909108
liste des actions:
[{'name': 'Share-XJMO', 'price': 9.39, 'profit': 39.98, 'returns': 3.754122}, {'name': 'Share-KMTG', 'price': 23.21, 'profit': 39.97, 'returns': 9.277037}, {'name': 'Share-MTLR', 'price': 16.49, 'profit': 39.97, 'returns': 6.591053},
{'name': 'Share-GTQK', 'price': 15.4, 'profit': 39.95, 'returns': 6.1523}, {'name': 'Share-LRBZ', 'price': 32.9, 'profit': 39.95, 'returns': 13.14355}, {'name': 'Share-WPLI', 'price': 34.64, 'profit': 39.91, 'returns': 13.824823999999
998}, {'name': 'Share-GIAJ', 'price': 10.75, 'profit': 39.9, 'returns': 4.28925}, {'name': 'Share-GHIZ', 'price': 28.0, 'profit': 39.89, 'returns': 11.1692}, {'name': 'Share-ZSDE', 'price': 15.11, 'profit': 39.88, 'returns': 6.025868}
, {'name': 'Share-IFCP', 'price': 29.23, 'profit': 39.88, 'returns': 11.656924000000002}, {'name': 'Share-FKJW', 'price': 21.08, 'profit': 39.78, 'returns': 8.385623999999998}, {'name': 'Share-NHWA', 'price': 29.18, 'profit': 39.77, '
returns': 11.604886000000002}, {'name': 'Share-LPDM', 'price': 39.35, 'profit': 39.73, 'returns': 15.633755}, {'name': 'Share-QQTU', 'price': 33.19, 'profit': 39.6, 'returns': 13.14324}, {'name': 'Share-USSR', 'price': 25.62, 'profit'
: 39.56, 'returns': 10.135272}, {'name': 'Share-EMOV', 'price': 8.89, 'profit': 39.52, 'returns': 3.513328000000001}, {'name': 'Share-LGWG', 'price': 31.41, 'profit': 39.5, 'returns': 12.40695}, {'name': 'Share-QLMK', 'price': 17.38,
'profit': 39.49, 'returns': 6.863362}, {'name': 'Share-SKKC', 'price': 24.87, 'profit': 39.49, 'returns': 9.821163}, {'name': 'Share-UEZB', 'price': 24.87, 'profit': 39.43, 'returns': 9.806241}, {'name': 'Share-CBNY', 'price': 1.22, '
profit': 39.31, 'returns': 0.479582}, {'name': 'Share-CGJM', 'price': 17.21, 'profit': 39.3, 'returns': 6.763529999999999}, {'name': 'Share-EVUW', 'price': 4.44, 'profit': 39.22, 'returns': 1.741368}, {'name': 'Share-FHZN', 'price': 6
.1, 'profit': 38.09, 'returns': 2.32349}, {'name': 'Share-MLGM', 'price': 0.01, 'profit': 18.86, 'returns': 0.0018859999999999999}]
 ---- Data set 2 ----

cout de la combinaison: 499.9800000000001
retours de la combinaison: 197.768345
profit de la combinaison: 39.5552512100484
liste des actions:
[{'name': 'Share-PATS', 'price': 27.7, 'profit': 39.97, 'returns': 11.07169}, {'name': 'Share-ALIY', 'price': 29.08, 'profit': 39.93, 'returns': 11.611643999999998
}, {'name': 'Share-JWGF', 'price': 48.69, 'profit': 39.93, 'returns': 19.441917}, {'name': 'Share-PLLK', 'price': 19.94, 'profit': 39.91, 'returns': 7.958054}, {'n
ame': 'Share-NDKR', 'price': 33.06, 'profit': 39.91, 'returns': 13.194246}, {'name': 'Share-FWBE', 'price': 18.31, 'profit': 39.82, 'returns': 7.291041999999999},
{'name': 'Share-LFXB', 'price': 14.83, 'profit': 39.79, 'returns': 5.900856999999999}, {'name': 'Share-ZOFA', 'price': 25.32, 'profit': 39.78, 'returns': 10.072296
}, {'name': 'Share-ANFX', 'price': 38.55, 'profit': 39.72, 'returns': 15.312059999999999}, {'name': 'Share-LXZU', 'price': 4.24, 'profit': 39.54, 'returns': 1.676496}, {'name': 'Share-FAPS', 'price': 32.57, 'profit': 39.54, 'returns': 12.878178}, {'name': 'Share-XQII', 'price': 13.42, 'profit': 39.51, 'returns': 5.302242}, {
'name': 'Share-ECAQ', 'price': 31.66, 'profit': 39.49, 'returns': 12.502534}, {'name': 'Share-JGTW', 'price': 35.29, 'profit': 39.43, 'returns': 13.914847}, {'name
': 'Share-IXCI', 'price': 26.32, 'profit': 39.4, 'returns': 10.37008}, {'name': 'Share-DWSK', 'price': 29.49, 'profit': 39.35, 'returns': 11.604315}, {'name': 'Sha
re-ROOM', 'price': 15.06, 'profit': 39.23, 'returns': 5.908038}, {'name': 'Share-VCXT', 'price': 29.19, 'profit': 39.22, 'returns': 11.448318}, {'name': 'Share-YFV
Z', 'price': 22.55, 'profit': 39.1, 'returns': 8.81705}, {'name': 'Share-OCKK', 'price': 3.16, 'profit': 36.39, 'returns': 1.149924}, {'name': 'Share-JMLZ', 'price
': 1.27, 'profit': 24.71, 'returns': 0.313817}, {'name': 'Share-DYVD', 'price': 0.28, 'profit': 10.25, 'returns': 0.0287}]
```

Set de donnée 1 :
Coût de la combinaison : 499,94€
Retours de la combinaisons:198,50€
Profits de la combinaison:39,70 %
Liste des actions:XJMO,KMTG,MTLR,GTQK,LRBZ,WPLI,GIAJ,GHIZ,ZSDE,IFCP,FKJW,-NHWA,LPDM,QQTU,USSR,EMOV,LGWG,QLMK,KKC,UEZB,CBNY,CGJM,EVUW,FHZN,MLGM

Set de donnée 2 :
Coût de la combinaison : 499,98€
Retours de la combinaisons:197,76€
Profites de la combinaison:39,55 %
Liste des actions:PATS,ALIY ,JWGF,PLLK,NDKR ,FWBE ,LFXB,ZOFA,ANFX,LXZU,FAPS,XQII,ECAQ,JGTW,IXCI,DWSK,ROOM,VCXT,YFVZ,OCKK,JMLZ,DYVD.

Solution optimisé : Greedy: performances :set de donnée 1

```
         10895 function calls in 0.011 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.010    0.010 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 codecs.py:260(__init__)
        5    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
     1002    0.002    0.000    0.004    0.000 csv.py:107(__next__)
        1    0.000    0.000    0.000    0.000 csv.py:81(__init__)
        1    0.000    0.000    0.000    0.000 csv.py:90(__iter__)
     2003    0.001    0.000    0.001    0.000 csv.py:93(fieldnames)
        1    0.000    0.000    0.010    0.010 main.py:30(main_set_1_opti_greedy2)
      956    0.000    0.000    0.000    0.000 opti_greedy.py:23(<lambda>)
      956    0.000    0.000    0.000    0.000 opti_greedy.py:27(calculate_return)
        2    0.000    0.000    0.000    0.000 opti_greedy.py:34(calculate_comb_cost)
        2    0.000    0.000    0.000    0.000 opti_greedy.py:43(calculate_comb_returns)
        1    0.000    0.000    0.000    0.000 opti_greedy.py:51(calculate_comb_profit)
        1    0.000    0.000    0.000    0.000 opti_greedy.py:59(greedy1)
        1    0.000    0.000    0.010    0.010 opti_greedy.py:76(main_greedy)
        1    0.002    0.002    0.002    0.002 opti_greedy.py:8(prepare_action_list)
        1    0.000    0.000    0.005    0.005 utils.py:4(extract_data)
        5    0.000    0.000    0.000    0.000 {built-in method _codecs.charmap_decode}
        1    0.000    0.000    0.000    0.000 {built-in method _csv.reader}
        1    0.000    0.000    0.011    0.011 {built-in method builtins.exec}
```

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    29     41.2 MiB     41.2 MiB           1   @profile
    30                                         def main_set_1_opti_greedy2():
    31     41.7 MiB      0.5 MiB           1       main_greedy(500, "E:\L7\Livrable\data\dataset1_Python+P7.csv")
```

Durée d'éxécution : 0,011sec
Utilisation mémoire : 41,7MiB
Mesures réalisées avec cProfile et memory_profiler

Solution optimisé : Greedy: performances :set de donnée 2

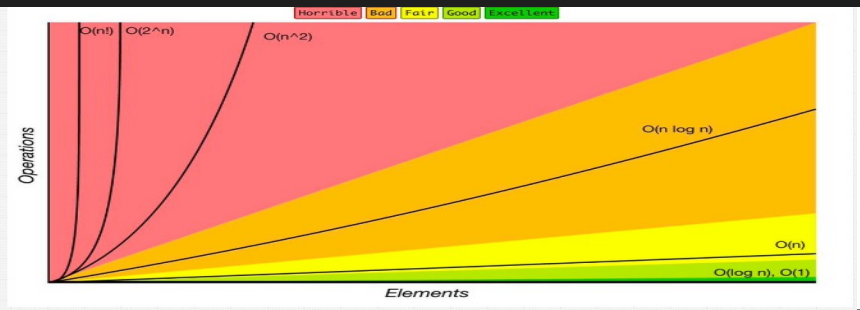
```
         9223 function calls in 0.008 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.008    0.008 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 codecs.py:260(__init__)
        4    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
     1001    0.002    0.000    0.004    0.000 csv.py:107(__next__)
        1    0.000    0.000    0.000    0.000 csv.py:81(__init__)
        1    0.000    0.000    0.000    0.000 csv.py:90(__iter__)
     2001    0.001    0.000    0.001    0.000 csv.py:93(fieldnames)
        1    0.000    0.000    0.008    0.008 main.py:41(main_set_2_opti_greedy1)
      541    0.000    0.000    0.000    0.000 opti_greedy.py:10(calculate_return)
        2    0.000    0.000    0.000    0.000 opti_greedy.py:17(calculate_comb_cost)
        2    0.000    0.000    0.000    0.000 opti_greedy.py:26(calculate_comb_returns)
        1    0.000    0.000    0.000    0.000 opti_greedy.py:34(calculate_comb_profit)
        1    0.001    0.001    0.002    0.002 opti_greedy.py:41(prepare_action_list)
      541    0.000    0.000    0.000    0.000 opti_greedy.py:56(<lambda>)
        1    0.000    0.000    0.000    0.000 opti_greedy.py:59(greedy1)
        1    0.000    0.000    0.008    0.008 opti_greedy.py:76(main_greedy)
        1    0.000    0.000    0.005    0.005 utils.py:4(extract_data)
        4    0.000    0.000    0.000    0.000 {built-in method _codecs.charmap_decode}
        1    0.000    0.000    0.000    0.000 {built-in method _csv.reader}
```

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    40     41.0 MiB     41.0 MiB           1   @profile
    41                                         def main_set_2_opti_greedy1():
    42     41.5 MiB      0.5 MiB           1       main_greedy(500, "E:\L7\Livrable\data\dataset2_Python+P7.csv")
```

Durée d'éxécution : 0,008sec
Utilisation mémoire : 41,5MiB
Mesures réalisées avec cProfile et memory_profiler

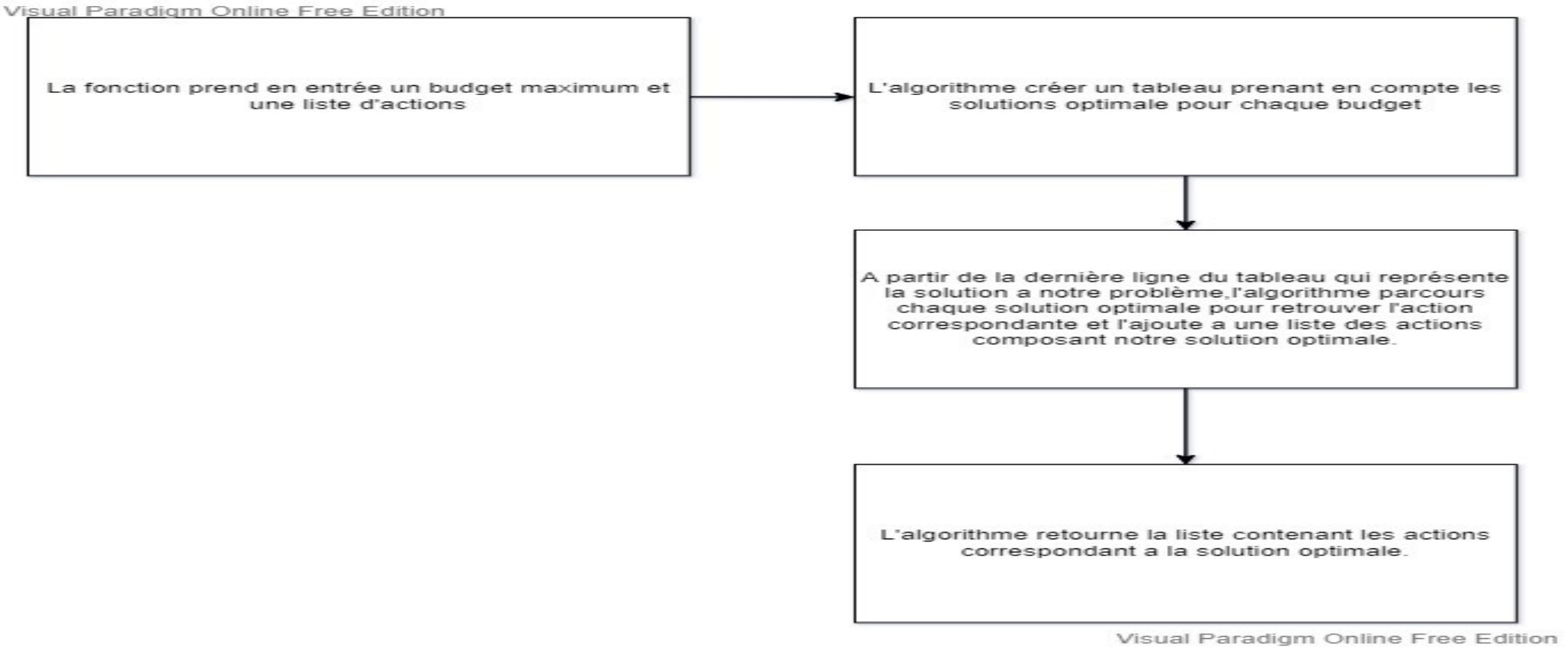
Solution optimisé : Greedy: Big-O

```python
def prepare_action_list(action_list: list):
    """Take the csv data as a list,create an empty list to store filtered data.
    Goes through the csv data, if the action price and profits are above 0,
    the action price and profit are converted to floats and the action return is added
    before adding the action to the filtered action list.
    Sort the filtered action list from worst[0] to best[-1] then returns it.
    """
    filtered_actions = []  # O(1)
    for action in action_list:  # O(n)
        if float(action["price"]) > 0 and float(action["profit"]) > 0:  # O(1)*4
            action["price"] = float(action["price"])  # O(1)*2
            action["profit"] = float(action["profit"])  # O(1)*2
            action["returns"] = calculate_return(action)  # O(1)
            filtered_actions.append(action)  # O(1)
    sorted_list = sorted(
        filtered_actions, key=lambda x: float(x["profit"]))  # O(n log(n))
    return sorted_list  # O(n log(n))

def greedy1(budget: int, action_list: list):
    """Take a budget,and a list of action from least[0] profitable
    to most[-1] profitable.Goes through the action list from the last index
    if the action price is under the available budget,it is added to
    the selected action list.
    Return the selected actions list when it went through the action list"""
    selected_actions = []  # O(1)
    total_price = 0  # O(1)
    while action_list:  # O(log(n))
        action = action_list.pop()  # O(1)
        if action["price"] + total_price <= budget:  # O(4)
            selected_actions.append(action)  # O(1)
            total_price += action["price"]  # O(1)
    return selected_actions  # O(log(n))
```



L'algorithme en lui même dénote une compléxité de O(log n),cependant le tri effectué avant est lui de O(n log(n)),la complexité totale de l'algorithme est donc de O(n Log(n))

Organigrame : solution dynamique :



Solution dynamique : résulats :

```
---- Data set 1 ----

cout de la combinaison: 499.94999999999993
retours de la combinaison: 198.546521
profit de la combinaison: 39.71327552755276
liste des actions:
[{'name': 'Share-MLGM', 'price': 0.01, 'profit': 18.86, 'returns': 0.0018859999999999999}, {'name': 'Share-KZBL', 'price': 28.99, 'profit': 39.14, 'returns': 11.346686}, {'name': 'Share-UEZB', 'price': 24.87, 'profit': 39.43, 'returns': 9.806241}, {'name': 'Share-QLMK', 'price': 17.38, 'profit': 39.49, 'returns': 6.863362}, {'name': 'Share-SKKC', 'price': 24.87, 'profit': 39.49, 'returns': 9.821163}, {'name': 'Share-LGWG', 'price': 31.41, 'profit': 39.5, 'returns': 12.40695}, {'name': 'Share-EMOV', 'price': 8.89, 'profit': 39.52, 'returns': 3.5133280000000001}, {'name': 'Share-USSR', 'price': 25.62, 'profit': 39.56, 'returns': 10.135272}, {'name': 'Share-QQTU', 'price': 33.19, 'profit': 39.6, 'returns': 13.14324}, {'name': 'Share-LPDM', 'price': 39.35, 'profit': 39.72, 'returns': 15.633755}, {'name': 'Share-NHWA', 'price': 29.18, 'profit': 39.77, 'returns': 11.604886000000002}, {'name': 'Share-FKJW', 'price': 21.08, 'profit': 39.78, 'returns': 8.385623999999998}, {'name': 'Share-ZSDE', 'price': 15.11, 'profit': 39.88, 'returns': 6.025868}, {'name': 'Share-IFCP', 'price': 29.23, 'profit': 39.88, 'returns': 11.656924000000002}, {'name': 'Share-GHIZ', 'price': 28.0, 'profit': 39.89, 'returns': 11.1692}, {'name': 'Share-WPLI', 'price': 34.64, 'profit': 39.9, 'returns': 13.824823999999998}, {'name': 'Share-GIAJ', 'price': 10.75, 'profit': 39.9, 'returns': 4.28925}, {'name': 'Share-GTQK', 'price': 15.4, 'profit': 39.95, 'returns': 6.1523}, {'name': 'Share-LRBZ', 'price': 32.9, 'profit': 39.95, 'returns': 13.14355}, {'name': 'Share-KMTG', 'price': 23.21, 'profit': 39.97, 'returns': 9.277037}, {'name': 'Share-MTLR', 'price': 16.48, 'profit': 39.97, 'returns': 6.591053}, {'name': 'Share-XJMO', 'price': 9.39, 'profit': 39.97, 'returns': 3.754122}]
 ---- Data set 2 ----

cout de la combinaison: 499.8999999999999
retours de la combinaison: 197.96466399999997
profit de la combinaison: 39.60085297059412
liste des actions:
[{'name': 'Share-SCWM', 'price': 6.42, 'profit': 38.1, 'returns': 2.44602}, {'name': 'Share-VCAX', 'price': 27.42, 'profit': 38.99, 'returns': 10.691058000000002}, {'name': 'Share-YFVZ', 'price': 22.55, 'profit': 39.1, 'returns': 8.81705}, {'name': 'Share-ROOM', 'price': 15.06, 'profit': 39.22, 'returns': 5.908038}, {'name': 'Share-DWSK', 'price': 29.49, 'profit': 39.35, 'returns': 11.604315}, {'name': 'Share-IXCI', 'price': 26.32, 'profit': 39.4, 'returns': 10.37008}, {'name': 'Share-JGTW', 'price': 35.29, 'profit': 39.43, 'returns': 13.914847}, {'name': 'Share-ECAQ', 'price': 31.66, 'profit': 39.49, 'returns': 12.502534}, {'name': 'Share-XQII', 'price': 13.42, 'profit': 39.51, 'returns': 5.302242}, {'name': 'Share-LXZU', 'price': 4.24, 'profit': 39.54, 'returns': 1.676496}, {'name': 'Share-FAPS', 'price': 32.57, 'profit': 39.54, 'returns': 12.878178}, {'name': 'Share-ANFX', 'price': 38.54, 'profit': 39.72, 'returns': 15.312059999999999}, {'name': 'Share-ZOFA', 'price': 25.32, 'profit': 39.78, 'returns': 10.072296}, {'name': 'Share-LFXB', 'price': 14.83, 'profit': 39.79, 'returns': 5.900856999999999}, {'name': 'Share-FWBE', 'price': 18.3, 'profit': 39.82, 'returns': 7.291041999999999}, {'name': 'Share-PLLK', 'price': 19.94, 'profit': 39.9, 'returns': 7.958054}, {'name': 'Share-NDKR', 'price': 33.06, 'profit': 39.9, 'returns': 13.194246}, {'name': 'Share-ALIY', 'price': 29.08, 'profit': 39.93, 'returns': 11.611643999999998}, {'name': 'Share-JWGF', 'price': 48.69, 'profit': 39.93, 'returns': 19.441917}, {'name': 'Share-PATS', 'price': 27.7, 'profit': 39.97, 'returns': 11.07169}]
```

Set de donnée 1 :
Coût de la combinaison : 499,94€
Retours de la combinaisons:198,54€
Profites de la combinaison:39,71 %
Liste des actions:MLGM,KZBL,SKKC,LGWG,EMOV,USSR,QQTU,LPDM,NHWA,FKJW,ZSDE,IFCP,GHIZ,WPLI,GIAJ,GTQK,LRBZ,KMTG,MTLR,XJMO

Set de donnée 2:
Coût de la combinaison : 499,89€
Retours de la combinaisons:197,96€
Profites de la combinaison:39,60 %
Liste des actions:SCWM,VCAX,YFVZ,ROOM,DWSK,IXCI,JGTW,ECAQ,XQII,LXZU,FAPS,ANFX,ZOFA,LFXB,FWBE,PLLK,NDKR,ALIY,JWGF,PATS,

Solution dynamique : mesures performance :

Set de donnée1 :

```
         45370465 function calls in 49.448 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   51.245   51.245 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 codecs.py:260(__init__)
        5    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
     1002    0.002    0.000    0.004    0.000 csv.py:107(__next__)
        1    0.000    0.000    0.000    0.000 csv.py:81(__init__)
        1    0.000    0.000    0.000    0.000 csv.py:90(__iter__)
     2003    0.001    0.000    0.001    0.000 csv.py:93(fieldnames)
        1    0.000    0.000   51.245   51.245 main.py:35(main_set_1_opti_dynamique)
        2    0.000    0.000    0.000    0.000 opti_knapsack.py:17(calculate_comb_returns)
        1    0.000    0.000    0.000    0.000 opti_knapsack.py:25(calculate_comb_profit)
        1    0.003    0.003    0.003    0.003 opti_knapsack.py:33(prepare_action_list)
      956    0.000    0.000    0.000    0.000 opti_knapsack.py:50(<lambda>)
        1   41.036   41.036   51.085   51.085 opti_knapsack.py:54(knapsack_dynamique)
        1    0.002    0.002    1.799    1.799 opti_knapsack.py:59(<listcomp>)
        2    0.000    0.000    0.000    0.000 opti_knapsack.py:8(calculate_comb_cost)
```

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    35     41.5 MiB     41.5 MiB           1   @profile
    36                                         def main_set_1_opti_dynamique():
    37     45.2 MiB      3.7 MiB           1       main_dynamique(500, "E:\L7\Livrable\data\dataset1_Python+P7.csv")
```

Durée d'éxécution : 49,448sec
Utilisation mémoire : 45,2MiB
Mesures réalisées avec cProfile et memory_profiler

Solution dynamique : mesures performance :

- Set de donnée 2 :

```
         25673403 function calls in 24.184 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   25.084   25.084 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 codecs.py:260(__init__)
        4    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
     1001    0.002    0.000    0.004    0.000 csv.py:107(__next__)
        1    0.000    0.000    0.000    0.000 csv.py:81(__init__)
        1    0.000    0.000    0.000    0.000 csv.py:90(__iter__)
     2001    0.001    0.000    0.001    0.000 csv.py:93(fieldnames)
        1    0.000    0.000   25.084   25.084 main.py:46(main_set_2_opti_dynamique)
        2    0.000    0.000    0.000    0.000 opti_knapsack.py:17(calculate_comb_returns)
        1    0.000    0.000    0.000    0.000 opti_knapsack.py:25(calculate_comb_profit)
        1    0.001    0.001    0.002    0.002 opti_knapsack.py:33(prepare_action_list)
      541    0.000    0.000    0.000    0.000 opti_knapsack.py:50(<lambda>)
        1   19.950   19.950   24.998   24.998 opti_knapsack.py:54(knapsack_dynamique)
        1    0.001    0.001    0.900    0.900 opti_knapsack.py:59(<listcomp>)
        2    0.000    0.000    0.000    0.000 opti_knapsack.py:8(calculate_comb_cost)
        1    0.000    0.000    0.000    0.000 opti_knapsack.py:81(return_value_to_normal)
        1    0.077    0.077   25.083   25.083 opti_knapsack.py:91(main_dynamique)
        1    0.000    0.000    0.004    0.004 utils.py:4(extract_data)
        4    0.000    0.000    0.000    0.000 {built-in method _codecs.charmap_decode}
        1    0.000    0.000    0.000    0.000 {built-in method _csv.reader}
        1    0.000    0.000   25.084   25.084 {built-in method builtins.exec}
```

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    45     41.3 MiB     41.3 MiB           1   @profile
    46                                         def main_set_2_opti_dynamique():
    47     42.8 MiB      1.5 MiB           1       main_dynamique(500, "E:\L7\Livrable\data\dataset2_Python+P7.csv")
```

Durée d'éxécution :
24,184sec
Utilisation mémoire :
42,8MiB
Mesures réalisées
avec cProfile et
memory_profiler

Solution dynamique : Big-O

```python
def knapsack_dynamique(raw_budget:int, action_list): #n= action_list;w=budget
    """create a matrice storing the best otpimal solution for each item and weigh available
    to obtain the best optimal combination,then goes back through the matrice to
    return the best combination of action possible as a list."""
    budget = raw_budget*100 #O(1)
    matrice = [[0 for x in range(budget + 1)]
               for x in range(len(action_list)+1)] #O(n*w)
    for i in range(1, len(action_list)+1):  #O(n)
        for w in range(1, budget+1):  #O(w)
            if action_list[i-1]["price"] <= w: #O(1)
                matrice[i][w] = max((action_list[i-1]["returns"])+matrice[i-1]
                                    [w-(action_list[i-1]["price"])], matrice[i-1][w]) #O(n)
            else:
                matrice[i][w] = matrice[i-1][w] #O(1)

    w = budget #O(1)
    n = len(action_list) #O(1)
    selected_actions = [] #O(1)
    while w >= 0 and n >= 0: #O(log(n))
        e = action_list[n-1] #O(1)
        if matrice[n][w] == matrice[n-1][w-e["price"]] + e["returns"]: #O(1)
            selected_actions.append(e) #O(1)
            w -= e["price"] #O(1)
        n -= 1 #O(1)
    return selected_actions
    #O(n*w)
```

Cet algorithme présente une complexité de O(n*w) où N correspond a l'ensemble de donnée et w au budget maximum alloué

Quel algorithme choisir ?

L’algorithme greedy :
-Extrêmement rapide (même avec un ensemble de donnée conséquent.
-Légèrement moins précis (les différences de retours sont de quelques centimes)

L’algorithme dynamique :
-Fourni la solution la plus optimisée
-Une complexité plus grande entraîne un temps d’éxécution croissant plus vite avec un ensemble de donnée plus grand.

Le choix de l’algorithme dépend donc de nos besoin : soit une solution plus optimisée,soit une qui fournira des résultats très proches de la solution optimisée très rapidement.Ma préférence se porte sur la solution dynamique,cependant cette dernière mettra plus de temps avec des ensemble de données plus grand.

Analyse des résultats :

Set de donnée 1 :

Résultats de Sienna : coût : 498,76€;retours:196,61€ taux de profit : 39,41 %

Résultats de l'algorithme greedy : coût 499,94€;retours :198,50€;profits :39,70 %

Résultats de l'algorithme dynamique : coût :499,94€,retours :198,54€;profits :39,71 %

Set de donnée 2 :

Résultats de Sienna : coût : 489,24€;retours:193,78€ taux de profit : 39.60%

Résultats de l'algorithme greedy : coût 499,98€;retours :197,76€;profits :39,70 %

Résultats de l'algorithme dynamique : coût :499,89€,retours :198,54€;profits :39,71 %

Dans les deux cas nous pouvons affirmer que les deux algorithmes produisent des résultats supérieurs aux investissements réalisés par sienna

Comment expliquer la différence :

- Les programmes traitent automatiquement les erreurs présentent dans les données.
- Les solutions optimisées sont composée de plusieurs actions proposant parfois des retours inférieurs a d’autres actions qui a première vue pourraient paraître plus intéressante.Cependant elles permettent de garder le budget nécessaire pour une combinaison plus intéressante.