③ ソートの比較

これまでの学習内容のまとめです。

ソートのまとめ

- これまでの学習内容をまとめると、以下のようになりました

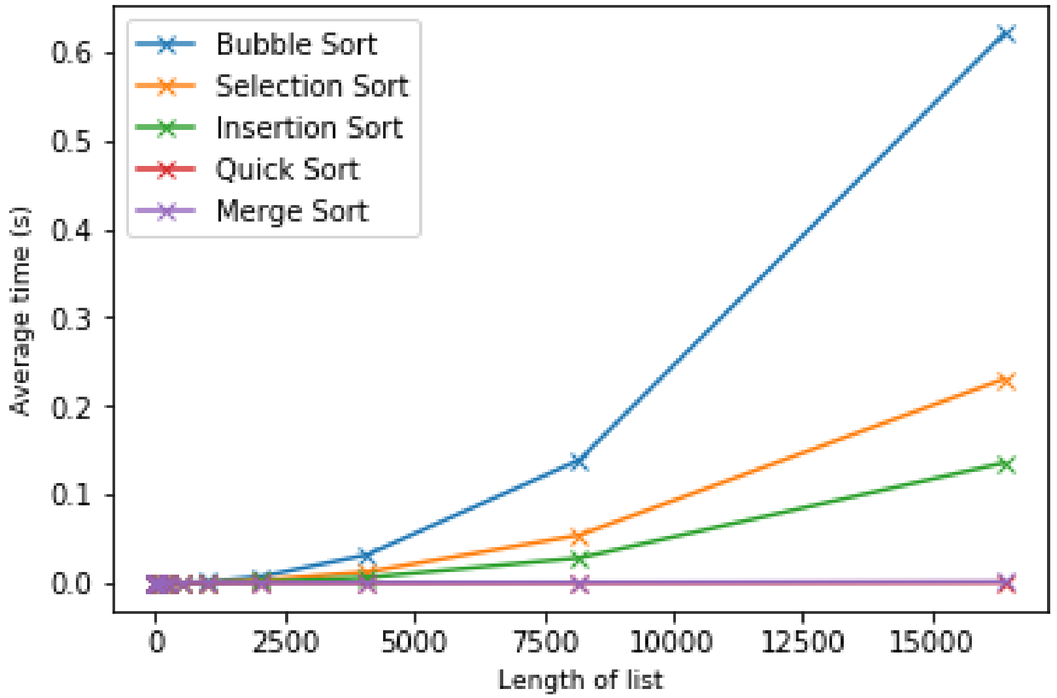
✔ Python のソート
- `sorted(iterable)` や `list.sort()` などの便利な機能を提供
- Pythonが提供するソートの計算量： ➡ $O(n \log n)$

✔ 単純な（直感的な）ソート
- 選択ソート、バブルソート、挿入ソート（どれも実用的ではない）
- 単純なソートの計算量： ➡ $O(n^2)$

✔ 効率的なソート
- クイックソート、マージソート
- 効率的なソートの平均計算量： ➡ $O(n \log n)$
- クイックソートの最悪計算量： ➡ $O(n^2)$

（参考）ソートの実行時間の比較

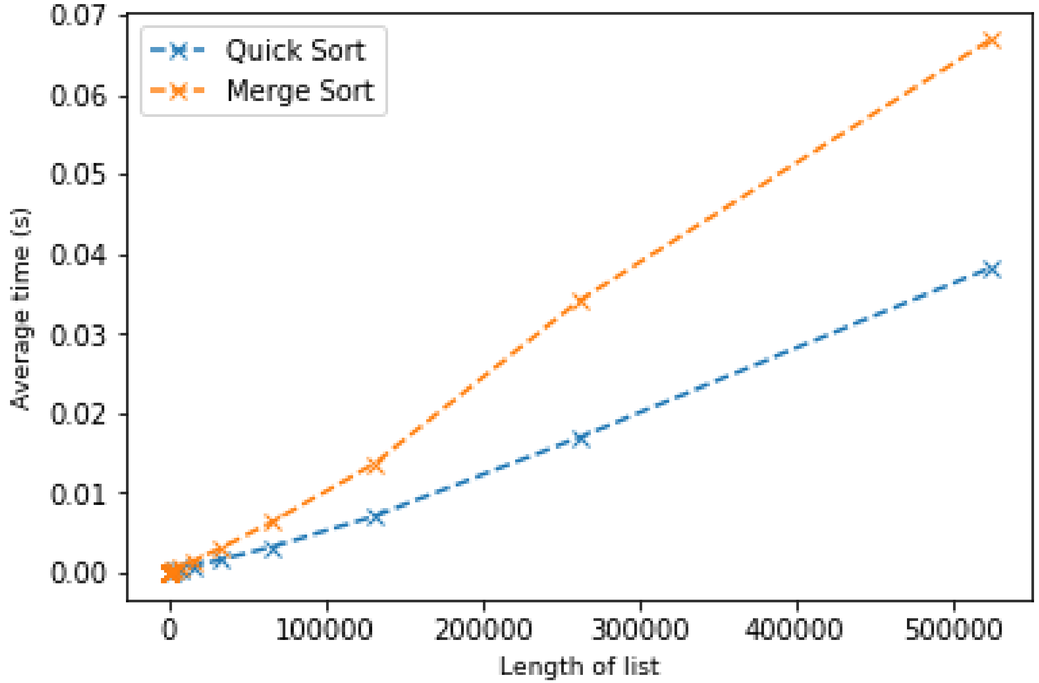
- 単純なソートアルゴリズムの実行時間を比較すると、多くの場合、以下のようになります
  - バブルソート ＞ 選択ソート ＞ 挿入ソート



C言語で実装したソートアルゴリズムの実行時間の比較例

（参考）ソートの実行時間の比較

- 効率的なソートアルゴリズムの実行時間を比較すると、多くの場合、クイックソートの方が早くなります
  - Pythonの実装でも、おそらく同様の傾向になっていると思います



C言語で実装したソートアルゴリズムの実行時間の比較例

（参考）ソートに関するその他の観点

- なお、この講義ではソートの計算量（実行時間）に注目をしましたが、他にもいくつかの観点があります
  - 普段意識することはないと思いますが、余裕がある人は覚えておきましょう
- メモリの使用量
  - 単純なソートである「選択ソート」は、リスト内でソートが完結します
  - 本来の「クイックソート」も、リスト内でソートが完結します
  - 一方で「マージソート」では、マージの過程で結果を入れるためのリストを、もう１つ用意することが必要になるため、メモリを多く利用します
- ソートの安定性
  - リストの中に「同等な要素」が複数出てきた場合に、ソート前の順序がソート後も保たれるようなソートを「安定なソート」と呼びます
  - いくつかのソートは安定ではありません（例えば、「マージソート」は安定ですが、本来の「クイックソート」は安定ではありません）

（参考）Excelのソート機能は「安定なソート」

- 例えば、Excelのソートは安定です
  - 以下の例では、点数でソートした後にクラスでソートしても、クラスの中では順番が点数でソートした状態になっている
- 実用的には、ソートが安定でないと困りますよね？

	A	B	C	D
1	学籍番号	氏名	クラス	点数
2	1	佐藤	1	70
3	2	鈴木	2	80
4	3	鈴木	1	90
5	4	田中	2	60
6	5	高橋	1	70
7	6	佐藤	2	65
8	7	鈴木	1	75
9	8	鈴木	2	85
10	9	田中	1	95
11	10	高橋	2	85

点数でソート
Sort by score

	A	B	C	D
1	学籍番号	氏名	クラス	点数
2	4	田中	2	60
3	6	佐藤	2	65
4	1	佐藤	1	70
5	5	高橋	1	70
6	7	鈴木	1	75
7	2	鈴木	2	80
8	8	鈴木	2	85
9	10	高橋	2	85
10	3	鈴木	1	90
11	9	田中	1	95

クラスでソート
Sorb by class

	A	B	C	D
1	学籍番号	氏名	クラス	点数
2	1	佐藤	1	70
3	5	高橋	1	70
4	7	鈴木	1	75
5	3	鈴木	1	90
6	9	田中	1	95
7	4	田中	2	60
8	6	佐藤	2	65
9	2	鈴木	2	80
10	8	鈴木	2	85
11	10	高橋	2	85