Sorting algorithms as in lectures

Key helper function

```python
def swap(unsorted, i, j):
    """
    Swap the i and j entries of the list unsorted.

    Parameters
    ----------
    unsorted : list or array
        List whose entries are to be swapped
    i, j : integer
        Indexes of entries to be swapped

    Returns
    -------
    sorted : list or array
        List with entries swapped
    """
    unsorted[i], unsorted[j] = unsorted[j], unsorted[i]
    return unsorted
```

Selection sort

```python
def selection_sort(unsorted):
    """
    Selection sort, sorting in non-decreasing order.

    Parameters
    ----------
    unsorted: list or array
        List to be sorted

    Returns
    -------
    sorted : list or array
        Sorted (in place) list
    """
    n = len(unsorted)
    for i in range(n-1):
        j = i
        for k in range(i+1, n):
            if unsorted[k] < unsorted[j]:
                j = k
        if unsorted[j] < unsorted[i]:
            unsorted = swap(unsorted, i, j)
    return unsorted
```

```
print(selection_sort([7, 2, 2, 6, 4]))
```

```
[2, 2, 4, 6, 7]
```

Insertion sort

```
def insertion_sort(unsorted):
    """
    Insertion sort, sorting in non-decreasing order.

    Parameters
    ----------
    unsorted: list or array
        List to be sorted

    Returns
    -------
    sorted : list or array
        Sorted (in place) list
    """
    n = len(unsorted)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if unsorted[j-1] > unsorted[j]:
                unsorted = swap(unsorted, j-1, j)
            else:
                break
    return unsorted
```

```
print(insertion_sort([7, 2, 2, 6, 4]))
```

```
[2, 2, 4, 6, 7]
```

Timing

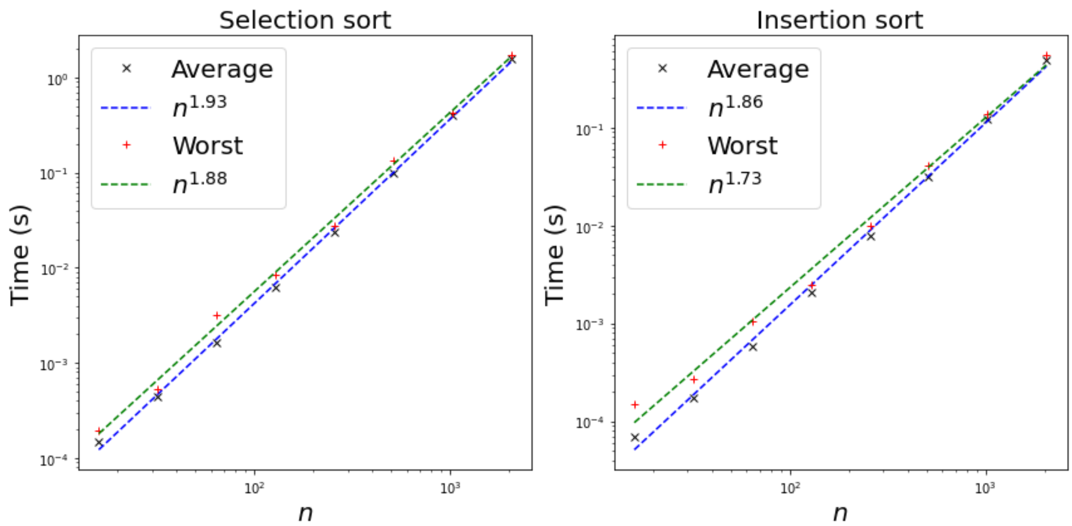
```
import numpy as np
from copy import deepcopy
import timeit
import matplotlib.pyplot as plt
import tqdm
```

```
ns = 2**np.arange(4, 12)
times_ave_ss = np.zeros(len(ns))
times_ave_is = np.zeros(len(ns))
times_max_ss = np.zeros(len(ns))
times_max_is = np.zeros(len(ns))
for i, n in enumerate(tqdm.tqdm(ns)):
    n_times_ss = np.zeros(100, dtype=np.float64)
    n_times_is = np.zeros(100, dtype=np.float64)
    for j in range(len(n_times_ss)):
        np.random.seed(j+100)
        n_times_ss[j] = timeit.timeit("selection_sort(unsorted)",
                                       setup=f"unsorted =
np.random.randint(0, {2*n}, size={n})",
                                       globals=globals(),
                                       number=5)
        np.random.seed(j+100)
        n_times_is[j] = timeit.timeit("insertion_sort(unsorted)",
                                       setup=f"unsorted =
np.random.randint(0, {2*n}, size={n})",
                                       globals=globals(),
                                       number=5)
    times_ave_ss[i] = n_times_ss.mean()
    times_ave_is[i] = n_times_is.mean()
    times_max_ss[i] = n_times_ss.max()
    times_max_is[i] = n_times_is.max()
```

```
100%|
 8/8 [04:36<00:00, 34.56s/it]
```

```
p_ave_ss = np.polyfit(np.log(ns), np.log(times_ave_ss), 1)
p_ave_is = np.polyfit(np.log(ns), np.log(times_ave_is), 1)
p_max_ss = np.polyfit(np.log(ns), np.log(times_max_ss), 1)
p_max_is = np.polyfit(np.log(ns), np.log(times_max_is), 1)
```

```
textsize=20
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
axes[0].loglog(ns, times_ave_ss, 'kx', label="Average")
axes[0].loglog(ns, np.exp(p_ave_ss[1])*ns**p_ave_ss[0], 'b--',
label=rf"$n^{{{p_ave_ss[0]:.2f}}}$")
axes[0].loglog(ns, times_max_ss, 'r+', label="Worst")
axes[0].loglog(ns, np.exp(p_max_ss[1])*ns**p_max_ss[0], 'g--',
label=rf"$n^{{{p_max_ss[0]:.2f}}}$")
axes[0].set_title("Selection sort", size=textsize)
axes[1].loglog(ns, times_ave_is, 'kx', label="Average")
axes[1].loglog(ns, np.exp(p_ave_is[1])*ns**p_ave_is[0], 'b--',
label=rf"$n^{{{p_ave_is[0]:.2f}}}$")
axes[1].loglog(ns, times_max_is, 'r+', label="Worst")
axes[1].loglog(ns, np.exp(p_max_is[1])*ns**p_max_is[0], 'g--',
label=rf"$n^{{{p_max_is[0]:.2f}}}$")
axes[1].set_title("Insertion sort", size=textsize)
for ax in axes:
    ax.legend(prop={'size': textsize})
    ax.set_xlabel(r"$n$", size=textsize)
    ax.set_ylabel("Time (s)", size=textsize)
fig.tight_layout()
```



We see that insertion sort is absolutely faster (the times at fixed n are lower than selection sort) and more efficient (the growth rate with n as measured by the best fit lines is smaller).