Let's make a model

Suppose that a population of some species in a well-defined area is currently 100000 individuals. Each year

- Approximately 2.5% of individuals give birth to one new individual;
- Approximately 1.5% of individuals die;
- Approximately 10000 individuals immigrate from outside the region to inside the region;
- Approximately 12000 individuals emigrate from inside the region to outside the region.

For each $n \in \mathbb{Z}_{\geq 0}$, let p_n denote the population of the region n years from now. From the information given we deduce:

$$p_0 = 100000; \text{ and}$$

$$\forall n \in \mathbb{Z}_{\geq 0} \quad p_{n+1} = p_n + \underbrace{.025 \times p_n}_{\text{babies}} - \underbrace{.015 \times p_n}_{\text{deaths}} + \underbrace{10000}_{\text{immigration}} - \underbrace{12000}_{\text{emigration}}$$

$$= p_n + .01 \times p_n - 2000$$

We have a model in the form of an implicit description of a sequence.

For each $n \in \mathbb{Z}_{\geq 0}$, let p_n denote the population of the region n years from now. From the information given we deduce:

$$p_0 = 100000; \text{ and}$$

$$\forall n \in \mathbb{Z}_{\geq 0} \quad p_{n+1} = p_n + \underbrace{.025 \times p_n}_{\text{babies}} - \underbrace{.015 \times p_n}_{\text{deaths}} + \underbrace{10000}_{\text{immigration}} - \underbrace{12000}_{\text{emigration}}$$

$$= p_n + .01 \times p_n - 2000$$

We have a model in the form of an implicit description of a sequence.

CHALLENGE: Find an explicit description of the sequence and prove that your answer is correct.

Sorting algorithms

Let $N \in \mathbb{N}$, S be a set, and $(x_n)_{n\in\{1,\dots,N\}} \subseteq S$.

Remember that this just means that $x_n \in S$ for each $n \in \{1, \dots, N\}$; it does not imply that all the x_n's are different.
So *sequences may contain some elements more than once.*

A **sorting algorithm** is a procedure for sorting a sequence into increasing order according to some specified ordering rule (*e.g.* numerical, alphabetical, etc.) *i.e.* it replaces $(x_n)_{n\in\{1,\dots,N\}}$ by a rearrangement $(y_n)_{n\in\{1,\dots,N\}}$ with

$$y_1 \leq y_2 \leq y_3 \cdots y_{N-1} \leq y_N$$

where "$\leq$" denotes the ordering rule.

Example:
$(x_n)_{n\in\{1,\dots,5\}}$ = Jane, Fred, Jo, Jane, Ann
$(y_n)_{n\in\{1,\dots,5\}}$ = Ann, Fred, Jane, Jane, Jo (in alphabetical order)

Sorting preliminaries

An **index set** I is a set of the form

$$I = \{i \in \{0, 1, 2, ...\} : s \leq i \leq f\} = \{s, \ldots, f\}$$

where $s, f \in \{0, 1, 2, ...\}$, $s \leq f$, are the **start index** and the **finish index**. **Example:** $I = \{3, 4, 5, 6\}$ $(s = 3, f = 6)$

For $I = \{s, \ldots, f\}$ we may denote the sequence $(a_n)_{n \in I}$ by $(a_n)_{s..f}$.

Example: Suppose $\forall n \in \mathbb{N}\ a_n = 2n + 1$. Then $(a_n)_{3..6}$ =7,9,11,13.

An **index permutation** on an index set I is a bijection (one-to-one correspondence) $\pi : I \to I$. For $I = \{s, \ldots, f\}$ the permutation can be specified using the notation

$$\pi = \begin{pmatrix} s & s+1 & \ldots & f \\ \pi(s) & \pi(s+1) & \ldots & \pi(f) \end{pmatrix}.$$

Example:

$\pi = \begin{pmatrix} 3 & 4 & 5 & 6 \\ 6 & 4 & 3 & 5 \end{pmatrix}$ means $\begin{array}{l} I = \{3, 4, 5, 6\} \\ \pi(3)=6,\ \pi(4)=4,\ \pi(5)=3,\ \pi(6)=5 \end{array}$.

More sorting preliminaries

Using index permutations for sorting has two benefits:

- it allows for more precise algorithm specification: and
- items being sorted do not get moved - only their indices are affected. This is valuable when the items have long and/or variable storage length.

A **reordering** of a sequence $(x_n)_{s..t}$ is a sequence $(y_n)_{s..t}$ where $y_n = x_{\pi(n)}$ for some index permutation π.
The reordering of a sequence $(a_n)_{s..t}$ can be donoted by $(a_{\pi(n)})_{s..t}$.

Example: For $\pi = \begin{pmatrix} 3 & 4 & 5 & 6 \\ 6 & 4 & 3 & 5 \end{pmatrix}$, if $(a_n)_{3..6} = 7,\ 9, 11, 13$. then $(a_{\pi(n)})_{3..6} = 13, 9,\ 7,\ 11$

Example: (names example recast using an index permutation)
If $(x_n)_{n\in\{1,\ldots,5\}}$ = Jane, Fred, Jo, Jane, Ann
then $(x_{\pi(n)})_{n\in\{1,\ldots,5\}}$ = Ann, Fred, Jane, Jane, Jo
where $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 1 & 4 & 3 \end{pmatrix}$ sorts the sequence into alphabetical order.

Least element algorithm

In writing algorithms from now on we will use the notation $a \leftarrow b$ to mean "assign a the value b, leaving b unchanged". (Some authors use $a := b$ for this.)

Input: Sequence $(x_i)_{s..f} \subseteq S$, an ordering rule "$\leq$" for S and an index function π on $\{s, \ldots, f\}$.

Output: Modification to π so that $x_{\pi(s)} \leq x_{\pi(i)}$ for $i = s, ..., f$.

Method:

$i \leftarrow s + 1$. [Initialisation]

$m \leftarrow s$, [m is a marker; $x_{\pi(m)}$ is the least sequence member so far tested]

Loop: If $i = f + 1$ stop.

If $x_{\pi(i)} < x_{\pi(m)}$ then $m \leftarrow i$.

$i \leftarrow i + 1$

Repeat loop

Swap the values of $\pi(s)$ and $\pi(m)$.

Example: $(s = 1,\ f = 6)$

	i	1	2	3	4	5	6
BEFORE	$\pi(i)$	1	2	3	4	5	6
	$x_{\pi(i)}$	F	D	C	E	B	C

Trace:

i	2	3	4	5	6	7
m	1	2	3	3	5	5
$x_{\pi(i)}$	D	C	E	B	C	-
$x_{\pi(m)}$	F	D	C	C	B	B

	i	1	2	3	4	5	6
AFTER	$\pi(i)$	**5**	2	3	4	**1**	6
	$x_{\pi(i)}$	**B**	D	C	E	**F**	C

Selection sort algorithm

Input: Sequence $(x_i)_{1..n} \subseteq S$, an ordering rule "$\leq$" for S and an index function π on $\{1, \ldots, n\}$.

Output: Modification to π, so that $(x_{\pi(i)})_{1..n}$ is in non-decreasing order $x_{\pi(1)} \leq x_{\pi(2)} \leq \cdots \leq x_{\pi(n)}$.

Method:

$s \leftarrow 1$ [Initialisation]

Loop: If $s = n$ stop.

Run least element algorithm on $(x_{\pi(i)})_{s..n}$

$s \leftarrow s + 1$

Repeat loop

Example:

	i:	1	2	3	4	5	6
Input: $(n = 6)$	$\pi(i)$	1	2	3	4	5	6
	$x_{\pi(i)}$	F	D	C	E	B	C
$s = 1$							
After 1st iteration	$\pi(i)$	**5**	2	3	4	**1**	6
	$x_{\pi(i)}$	**B**	D	C	E	**F**	C
$s = 2$							
After 2nd iteration	$\pi(i)$	5	**3**	**2**	4	1	6
	$x_{\pi(i)}$	B	**C**	**D**	E	F	C
$s = 3$							
After 3rd iteration	$\pi(i)$	5	3	**6**	4	1	**2**
	$x_{\pi(i)}$	B	C	**C**	E	F	**D**
$s = 4$							
After 4th iteration	$\pi(i)$	5	3	6	**2**	1	**4**
	$x_{\pi(i)}$	B	C	C	**D**	F	**E**
$s = 5$							
After final iteration	$\pi(i)$	5	3	6	2	**4**	**1**
	$x_{\pi(i)}$	B	C	C	D	**E**	**F**

Selection sort: number of operations

How many operations are required by Selection Sort?

By *operation* here we mean any comparison step;

i.e. a step of the form "If $x_{\pi(i)} \leq x_{\pi(j)}$ then ..."

The loop of the Selection Sort algorithm is iterated $n-1$ times; once each for $s = 1, \ldots, n-1$.

Iteration s runs the Least Element algorithm on $(x_{\pi(i)})_{s..n}$ and so uses $n-s$ comparisons.

So:

1st	iteration	uses	$n-1$	comparisons
2nd	iteration	uses	$n-2$	comparisons
$\vdots$	$\vdots$		$\vdots$	$\vdots$
last	iteration	uses	1	comparison

Hence the total number of comparisons, T_n say, is given by

$$1 + 2 + \cdots + (n-1) = (n-1)\left(\frac{1+(n-1)}{2}\right) \text{ (sum of an arithmetic series).}$$

That is: $\forall \boldsymbol{n \in \mathbb{N}} \quad \boldsymbol{T_n = \frac{n(n-1)}{2}}.$

Other sorting algorithms

There are many different sorting algorithms, with various pros and cons. A full study of the topic belongs in course on algorithms and data structures.

We will look at just one more; "Merge Sort".
This will provide us with an opportunity to compare two algorithms designed to do the same job – what are their respective advantages and disadvantages?

In order to keep the description simple, I will not use an indexing function π in specifying the algorithm, though it is possible, and often preferable, to do so.

As with Selection Sort, Merge Sort makes use of a sub-algorithm, which we treat first.

Merge algorithm

Input: Two lists (sequences) $(a_i)_{i\in\{1,\ldots,n\}} \subseteq S$ and $(b_j)_{j\in\{1,\ldots,p\}} \subseteq S$ pre-sorted according to an ordering rule "$\leq$" on S.

Output: In-order list (sorted sequence) $(z_k)_{k\in\{1,\ldots,n+p\}}$ that merges the two input lists.

Method:

$i, j, k \leftarrow 1$. [Initialize the indeces for the a-, b- and z- lists]

Loop: If $k = n+p+1$ stop. [there will be $n+p$ items in the z-list]

If $i = n+1$ then $[\, z_k \leftarrow b_j,\ j \leftarrow j+1 \,]$ [a-list empty; take from b-list]

Else if $j = p+1$ then $[\, z_k \leftarrow a_i,\ i \leftarrow i+1 \,]$ [b-list empty; take from a-list]

Else if $a_i < b_j$ then $[\, z_k \leftarrow a_i,\ i \leftarrow i+1 \,]$ [a-list item less; take it]

Else $[\, z_k \leftarrow b_j,\ j \leftarrow j+1 \,]$ [else take item from b-list]

$k \leftarrow k+1$ [prepare to add next item to z-list]

Repeat loop.

Merge algorithm: example of execution

Example: Merge $(1, 3, 7)$ and $(2, 3, 6, 8, 9)$.

After iteration	i	j	k	a_i [green]	b_j [green]	$(z_1, \ldots, z_{k-1})$
0	1	1	1	$(\mathbf{1}, 3, 7)$	$(\mathbf{2}, 3, 6, 8, 9)$	$()$
1	2	1	2	$(1, \mathbf{3}, 7)$	$(\mathbf{2}, 3, 6, 8, 9)$	(1)
2	2	2	3	$(1, \mathbf{3}, 7)$	$(2, \mathbf{3}, 6, 8, 9)$	$(1, 2)$
3	2	3	4	$(1, \mathbf{3}, 7)$	$(2, 3, \mathbf{6}, 8, 9)$	$(1, 2, 3)$
4	3	3	5	$(1, 3, \mathbf{7})$	$(2, 3, \mathbf{6}, 8, 9)$	$(1, 2, 3, 3)$
5	3	4	6	$(1, 3, \mathbf{7})$	$(2, 3, 6, \mathbf{8}, 9)$	$(1, 2, 3, 3, 6)$
6	4	4	7	$(1, 3, 7)$	$(2, 3, 6, \mathbf{8}, 9,)$	$(1, 2, 3, 3, 6, 7)$
7	4	5	8	$(1, 3, 7)$	$(2, 3, 6, 8, \mathbf{9})$	$(1, 2, 3, 3, 6, 7, 8)$
8	4	6	9	$(1, 3, 7)$	$(2, 3, 6, 8, 9)$	$(1, 2, 3, 3, 6, 7, 8, 9)$

Merge Sort algorithm

Input: $r \in \mathbb{N}$, $(x_n)_{n\in\{1,\ldots,2^r\}} \subseteq S$,and an ordering rule "$\leq$" for S.
(For lists whose length is not a power of 2, see workshop question.)

Output: In-order list (sorted sequence) $(z_n)_{n\in\{1,\ldots,2^r\}}$ that is a rearrangement of the input list.

Method: There are r steps.
If $r < 3$ adjust the description below accordingly.

- Step 1: Apply the Merge algorithm 2^{r-1} times with inputs $\{(x_1),(x_2)\}$, $\{(x_3),(x_4)\}$,...,$\{(x_{2^r-1}),(x_{2^r})\}$.
This gives 2^{r-1} in-order lists of length 2.
- Step 2: Apply the Merge algorithm 2^{r-2} times with pairs of these lists as input.
This gives 2^{r-2} in-order lists of length $2 \times 2 = 2^2$.
- Steps 3 to r: Continue in this vein until you have just one $(= 2^{r-r})$ in-order list with 2^r elements.

Merge sort: example

Example: merge sort $(1, 2, 6, 1, 7, 9, 4, 5)$.

Step 1:
Merging $\{(1), (2)\}$ gives $(1, 2)$.
Merging $\{(6), (1)\}$ gives $(1, 6)$.
Merging $\{(7), (9)\}$ gives $(7, 9)$.
Merging $\{(4), (5)\}$ gives $(4, 5)$.

Step 2:
Merging $\{(1, 2), (1, 6)\}$ gives $(1, 1, 2, 6)$.
Merging $\{(7, 9), (4, 5)\}$ gives $(4, 5, 7, 9)$.

Step 3:
Merging $\{(1, 1, 2, 6), (4, 5, 7, 9)\}$ gives $(1, 1, 2, 4, 5, 6, 7, 9)$.

Merge sort: counting comparisons

As with Selection Sort, we can analyze the complexity of Merge sort by counting the number of comparisons involved.

Revisiting the previous example, merge sort $(1, 2, 6, 1, 7, 9, 4, 5)$:

(1)	(2)	(6)	(1)	(7)	(9)	(4)	(5)	
$(1,2)$		$(1,6)$		$(7,9)$		$(4,5)$		$1+1+1+1=4$ comps
	$(1,1,2,6)$				$(4,5,7,9)$			$3+2=5$ comps
			$(1,1,2,4,5,6,7,9)$					6 comps
							TOTAL:	15 comparisons

Note for example that when merging $(7, 9)$ and $(4, 5)$ only 2 comparisons are used:
7 and 4 are compared; 4 is transferred
7 and 5 are compared; 5 is transferred
7 and 9 are transferred without comparison (other list exhausted.)

Merge sort: number of operations

Since the number of comparisons used to Merge Sort a list of length n depends to some extent on the nature of the list, there is no precise formula for this number as there is with Selection Sort.

However an upper bound is given by the number of *transfers*.

Let T_r denote the number of transfers required to sort a sequence of length 2^r using Merge sort.

Now $2^{r-1} + 2^{r-1} = 2^r$ transfers are required for the Merge algorithm to merge two sequences of length 2^{r-1}, so an implicit definition for T_r is $\begin{cases} \boldsymbol{T_r = 2^r + 2\,T_{r-1}} & \boldsymbol{\forall r \in \mathbb{N}\backslash\{1\}} \\ \boldsymbol{T_1 = 2.} \end{cases}$

So $T_1 = 2, \quad T_2 = 2^2 + 4 = 8, \quad T_3 = 2^3 + 2{\times}8 = 3{\times}2^3,$

$T_4 = 2^4 + 2 \times (3{\times}2^3) = 4{\times}2^4, \quad \ldots$

Claim: $\boldsymbol{\forall r \in \mathbb{N} \quad T_r = r2^r}$. Verify by induction!

(The sequence $(T_r)_{r\in\mathbb{N}}$ is neither geometric, arithmetic nor mixed.)

Merge Sort and Selection Sort compared

Merge Sort sorts a sequence of length 2^r with less than $r2^r$ comparisons.

For the same length, Selection Sort uses $\dfrac{2^r(2^r-1)}{2}$ comparisons.

Merge sort is **much faster**, e.g. for $r = 10$ (so $N = 1024$):

$$\begin{aligned} \text{Merge sort:} \quad r2^r &= 10\,240 \\ \text{Selection sort:} \quad \frac{2^r(2^r-1)}{2} &= 523\,776. \end{aligned}$$

Merge sort can be modified to work even faster on machines with parallel processing capabilities since then many of the uses of the Merge algorithm can be done simultaneously. There is no direct way to use parallel processing with Selection sort.

Selection Sort may be simpler to program, especially if indexing is required to avoid transfers of large blocks of data.
For short lists on high speed computers, slower speed may not matter.

END OF SECTION B2