Discrete Mathematical Models

Lecture 13

Kane Townsend

Semester 2, 2024

Collaboration Statement for Assignment 1

I have posted an example collaboration statement on the Wattle page. For example:

The assignment I have submitted has been produced according to the rules on Page 1 of the MATH?005 Assignment 1. The solutions produced have been written/typed on my own. I have not used AI to prepare my solutions. I have provided references to the sources I have used where appropriate.

Student ID:

Sorting (Cont.)

Merge Algorithm

The **merge algorithm** takes two sequences in S, already ordered according to some $\leq$ and outputs an ordered sequence that merges the two input sequences.

Example: Merge $(x_i)_{1..2} = (1, 4)$ and $(y_j)_{1..3} = (2, 3, 5)$.

Compare $x_1 = 1$ and $y_1 = 2$. Since $1 < 2$ we assign $z_1 = x_1 = 1$.

Compare $x_2 = 4$ and $y_1 = 2$. Since $2 < 4$, we assign $z_2 = y_1 = 2$.

Compare $x_2 = 4$ and $y_2 = 3$. Since $3 < 4$, we assign $z_3 = y_2 = 3$.

Compare $x_2 = 4$ and $y_3 = 5$. Since $4 < 5$, we assign $z_4 = x_2 = 4$.

There is no x_3 so we assign $z_5 = y_3 = 5$.

We have reached the end of both $(x_i)_{1..2}$ and $(y_j)_{1..3}$. So we end the merging and we have $(1, 2, 3, 4, 5)$ as our merged list.

Merge algorithm

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

Merge algorithm

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

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

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 indices for the a-, b- and z- lists]

Merge algorithm

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

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

Method:

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

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

Merge algorithm

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

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

Method:

$i, j, k \leftarrow 1$. [Initialize the indices 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]

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 indices 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]

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 indices 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]

Merge algorithm

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

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

Method:

$i, j, k \leftarrow 1$. [Initialize the indices 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]

Merge algorithm

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

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

Method:

$i, j, k \leftarrow 1$. [Initialize the indices 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 $(a_i)_{1..3} = (1,3,7)$ and $(b_j)_{1..5} = (2,3,6,8,9)$.

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

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

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

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

After iteration	i	j	k	a_i	b_j	$(z_1, \dots, z_{k-1})$
0	1	1	1	$(\color{red}{1}, 3, 7)$	$(\color{red}{2}, 3, 6, 8, 9)$	$()$
1	2	1	2	$(1, \color{red}{3}, 7)$	$(\color{red}{2}, 3, 6, 8, 9)$	(1)
2	2	2	3	$(1, \color{red}{3}, 7)$	$(2, \color{red}{3}, 6, 8, 9)$	$(1, 2)$

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

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

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

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

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

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

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

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

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

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

Merge algorithm: example of execution

Example: Merge $(a_i)_{1..3} = (1, 3, 7)$ and $(b_j)_{1..5} = (2, 3, 6, 8, 9)$.

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

Merge Sort Algorithm

Consider the sequence $(4, 2, 3, 1)$.

Sort this sequence into increasing order.

Apply merge algorithm for $\{(4), (2)\}$ and $\{(3), (1)\}$.

This gives $(2, 4)$ and $(1, 3)$.

Now apply the merge algorithm on $\{(2, 4), (1, 3)\}$.

This gives $(1, 2, 3, 4)$.

Merge Sort algorithm

Input: $r \in \mathbb{N},\ (x_n)_{n\in\{1,\dots,2^r\}} \subseteq S$,and an ordering rule "$\leq$" for S.

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.

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.

Merge Sort algorithm

Input: $r \in \mathbb{N}$, $(x_n)_{n\in\{1,\dots,2^r\}} \subseteq S$,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) $(z_n)_{n\in\{1,\dots,2^r\}}$ 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 $2 \times 2 = 2^2$.

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 $(1, 2, 6, 1, 7, 9, 4, 5)$.

Merge sort: example

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

Step 1:
Merging $\{(1), (2)\}$ gives $(1, 2)$.

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 $(4, 5)$.

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 $(4, 5, 7, 9)$.

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.

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 $(1, 2, 6, 1, 7, 9, 4, 5)$:

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 $(1, 2, 6, 1, 7, 9, 4, 5)$:

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

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:

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

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 $(1, 2, 6, 1, 7, 9, 4, 5)$:

$(1)\quad(2)\quad(6)\quad(1)\quad(7)\quad(9)\quad(4)\quad(5)$	
$(1, 2)\quad(1, 6)\quad(7, 9)\quad(4, 5)$	$1+1+1+1=4$ comps
$(1, 1, 2, 6)\quad(4, 5, 7, 9)$	$3+2=5$ 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

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 $(1, 2, 6, 1, 7, 9, 4, 5)$:

$$
\begin{array}{ll}
(1) \quad (2) \quad (6) \quad (1) \quad (7) \quad (9) \quad (4) \quad (5) & \\
(1,2) \qquad (1,6) \qquad (7,9) \qquad (4,5) & 1{+}1{+}1{+}1{=}4 \text{ comps} \\
(1,1,2,6) \qquad\qquad (4,5,7,9) & 3{+}2{=}5 \text{ comps} \\
(1,1,2,4,5,6,7,9) & 6 \text{ comps} \\
\text{TOTAL:} & 15 \text{ comparisons}
\end{array}
$$

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.

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*.

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.

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},

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

$$\begin{cases} \boldsymbol{T_r} = 2^{\boldsymbol{r}} + 2\boldsymbol{T_{r-1}} & \forall \boldsymbol{r} \in \mathbb{N}\backslash\{1\} \\ \boldsymbol{T}_1 = 2. \end{cases}$$

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 $\begin{cases} \boldsymbol{T_r = 2^r + 2T_{r-1}} & \forall \boldsymbol{r} \in \mathbb{N}\backslash\{1\} \\ \boldsymbol{T_1 = 2.} \end{cases}$

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

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

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 $\begin{cases} \boldsymbol{T_r = 2^r + 2T_{r-1}} \quad \forall \boldsymbol{r} \in \mathbb{N}\backslash\{1\} \\ \boldsymbol{T_1 = 2.} \end{cases}$

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: $\forall \boldsymbol{r} \in \mathbb{N} \quad \boldsymbol{T_r = r2^r}$.

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 $\begin{cases} \boldsymbol{T_r = 2^r + 2T_{r-1}} & \forall \boldsymbol{r} \in \mathbb{N}\backslash\{1\} \\ \boldsymbol{T_1 = 2.} \end{cases}$

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: $\forall \boldsymbol{r} \in \mathbb{N} \quad \boldsymbol{T_r = r2^r}$. 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 at most $r2^r$ comparisons.

Merge Sort and Selection Sort compared

Merge Sort sorts a sequence of length 2^r with at most $r2^r$ comparisons.

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

Merge Sort and Selection Sort compared

Merge Sort sorts a sequence of length 2^r with at most $r2^r$ 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 $N = 1024$):

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

Merge Sort and Selection Sort compared

Merge Sort sorts a sequence of length 2^r with at most $r2^r$ 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 $N = 1024$):

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

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.

Merge Sort and Selection Sort compared

Merge Sort sorts a sequence of length 2^r with at most $r2^r$ 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 $N = 1024$):

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

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.

Merge Sort and Selection Sort compared

Merge Sort sorts a sequence of length 2^r with at most $r2^r$ 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 $N = 1024$):

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

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.

B3: Matrices

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets.

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets. The expression "$m \times n$" describes the **shape** of the matrix.

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets. The expression "$m \times n$" describes the **shape** of the matrix. Examples of various shapes are:

$$\mathrm{A} = \begin{bmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \end{bmatrix} \qquad \mathrm{B} = \begin{bmatrix} \pi/2 \\ -\pi/2 \end{bmatrix} \qquad \mathrm{C} = \begin{pmatrix} \frac{1}{5} & \frac{2}{5} & \frac{2}{5} \end{pmatrix}$$

A is a 2 $\times$ 3 matrix over $\mathbb{Z}$ **B is a 2 $\times$ 1 matrix over $\mathbb{R}$** **C is a 1 $\times$ 3 matrix over $\mathbb{Q}$**

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets. The expression "$m \times n$" describes the **shape** of the matrix. Examples of various shapes are:

$$\mathrm{A} = \underbrace{\begin{bmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \end{bmatrix}}_{\text{A is a } 2 \times 3 \text{ matrix over } \mathbb{Z}} \qquad \mathrm{B} = \underbrace{\begin{bmatrix} \pi/2 \\ -\pi/2 \end{bmatrix}}_{\text{B is a } 2 \times 1 \text{ matrix over } \mathbb{R}} \qquad \mathrm{C} = \underbrace{\begin{pmatrix} \frac{1}{5} & \frac{2}{5} & \frac{2}{5} \end{pmatrix}}_{\text{C is a } 1 \times 3 \text{ matrix over } \mathbb{Q}}$$

The set of all $m \times n$ matrices over S is denoted by $M_{m\times n}(S)$, so

$$\mathrm{A} \in M_{2\times 3}(\mathbb{Z}), \qquad \mathrm{B} \in M_{2\times 1}(\mathbb{R}), \qquad \mathrm{C} \in M_{1\times 3}(\mathbb{Q}).$$

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets. The expression "$m \times n$" describes the **shape** of the matrix. Examples of various shapes are:

$$\mathrm{A} = \begin{bmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \end{bmatrix} \qquad \mathrm{B} = \begin{bmatrix} \pi/2 \\ -\pi/2 \end{bmatrix} \qquad \mathrm{C} = \begin{pmatrix} \frac{1}{5} & \frac{2}{5} & \frac{2}{5} \end{pmatrix}$$

A is a 2×3 matrix over $\mathbb{Z}$ **B is a 2×1 matrix over $\mathbb{R}$** **C is a 1×3 matrix over $\mathbb{Q}$**

The set of all $m \times n$ matrices over S is denoted by $M_{m\times n}(S)$, so

$$\mathrm{A} \in M_{2\times 3}(\mathbb{Z}), \qquad \mathrm{B} \in M_{2\times 1}(\mathbb{R}), \qquad \mathrm{C} \in M_{1\times 3}(\mathbb{Q}).$$

Matrices with an equal number of rows and columns, *i.e.* $m = n$, play a particularly important role in mathematics.

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets. The expression "$m \times n$" describes the **shape** of the matrix. Examples of various shapes are:

$$\mathrm{A} = \begin{bmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \end{bmatrix} \qquad \mathrm{B} = \begin{bmatrix} \pi/2 \\ -\pi/2 \end{bmatrix} \qquad \mathrm{C} = \begin{pmatrix} \frac{1}{5} & \frac{2}{5} & \frac{2}{5} \end{pmatrix}$$

A is a 2×3 matrix over $\mathbb{Z}$ **B is a 2×1 matrix over $\mathbb{R}$** **C is a 1×3 matrix over $\mathbb{Q}$**

The set of all $m \times n$ matrices over S is denoted by $M_{m \times n}(S)$, so

$$\mathrm{A} \in M_{2\times 3}(\mathbb{Z}), \qquad \mathrm{B} \in M_{2\times 1}(\mathbb{R}), \qquad \mathrm{C} \in M_{1\times 3}(\mathbb{Q}).$$

Matrices with an equal number of rows and columns, *i.e.* $m = n$, play a particularly important role in mathematics.

For such a **square matrix** M over S we simply write $\mathrm{M} \in M_n(S)$.

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets. The expression "$m \times n$" describes the **shape** of the matrix. Examples of various shapes are:

$$\mathrm{A} = \begin{bmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \end{bmatrix} \qquad \mathrm{B} = \begin{bmatrix} \pi/2 \\ -\pi/2 \end{bmatrix} \qquad \mathrm{C} = \begin{pmatrix} \frac{1}{5} & \frac{2}{5} & \frac{2}{5} \end{pmatrix}$$

A is a 2×3 matrix over $\mathbb{Z}$ **B is a 2×1 matrix over $\mathbb{R}$** **C is a 1×3 matrix over $\mathbb{Q}$**

The set of all $m \times n$ matrices over S is denoted by $M_{m\times n}(S)$, so

$$\mathrm{A} \in M_{2\times 3}(\mathbb{Z}), \qquad \mathrm{B} \in M_{2\times 1}(\mathbb{R}), \qquad \mathrm{C} \in M_{1\times 3}(\mathbb{Q}).$$

Matrices with an equal number of rows and columns, *i.e.* $m = n$, play a particularly important role in mathematics.

For such a **square matrix** M over S we simply write $\mathrm{M} \in M_n(S)$.

Examples: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \in M_2(\mathbb{N}),$

What is a matrix (plural: matrices) ?

Definition: Let S be a set, and $m, n \in \mathbb{N}$.

An $m \times n$ **matrix** (over S) is a rectangular array of members of S, the array having m rows and n columns. The array is enclosed left and right with parentheses or brackets. The expression "$m \times n$" describes the **shape** of the matrix. Examples of various shapes are:

$$\mathrm{A} = \underbrace{\begin{bmatrix} 1 & -2 & 3 \\ -4 & 5 & -6 \end{bmatrix}}_{\textbf{A is a } 2 \times 3 \textbf{ matrix over } \mathbb{Z}} \qquad \mathrm{B} = \underbrace{\begin{bmatrix} \pi/2 \\ -\pi/2 \end{bmatrix}}_{\textbf{B is a } 2 \times 1 \textbf{ matrix over } \mathbb{R}} \qquad \mathrm{C} = \underbrace{\begin{pmatrix} \frac{1}{5} & \frac{2}{5} & \frac{2}{5} \end{pmatrix}}_{\textbf{C is a } 1 \times 3 \textbf{ matrix over } \mathbb{Q}}$$

The set of all $m \times n$ matrices over S is denoted by $M_{m\times n}(S)$, so

$$\mathrm{A} \in M_{2\times 3}(\mathbb{Z}), \qquad \mathrm{B} \in M_{2\times 1}(\mathbb{R}), \qquad \mathrm{C} \in M_{1\times 3}(\mathbb{Q}).$$

Matrices with an equal number of rows and columns, *i.e.* $m = n$, play a particularly important role in mathematics.

For such a **square matrix** M over S we simply write $\mathrm{M} \in M_n(S)$.

Examples: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \in M_2(\mathbb{N})$, $\qquad \begin{pmatrix} a & b & c \\ c & a & b \\ b & c & a \end{pmatrix} \in M_3(\{a, b, c\})$.

Indexing

A generic member of $M_{m \times n}(S)$ is written

$$\mathsf{A} = (a_{i,j}) = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix}$$

so that $a_{i,j} \in S$ denotes the entry in row i, column j, of A.

Indexing

A generic member of $M_{m\times n}(S)$ is written

$$\mathrm{A} = (a_{i,j}) = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix}$$

so that $a_{i,j} \in S$ denotes the entry in row i, column j, of A.

NB: The **row index** i always comes *before* the **column index** j.

Indexing

A generic member of $M_{m\times n}(S)$ is written

$$A = (a_{i,j}) = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{pmatrix}$$

so that $a_{i,j} \in S$ denotes the entry in row i, column j, of A.

NB: The **row index** i always comes *before* the **column index** j.

Example: For the matrix $A = \begin{bmatrix} 2 & 7 \\ 0 & -3 \end{bmatrix}$ we have

$$a_{1,1} = 2, \qquad a_{1,2} = 7, \qquad a_{2,1} = 0, \qquad a_{2,2} = -3.$$

Two dimensional information

Let S be a set, $n \in \mathbb{N}$.

Two dimensional information

Let S be a set, $n \in \mathbb{N}$. Elements of $S^n = \underbrace{S \times S \times \cdots \times S}_{n \text{ copies of } S}$ correspond to sequences $(a_j)_{1..n}$, i.e. functions

$$\begin{aligned} a : \{1, ..., n\} &\to S \\ j &\mapsto a_j. \end{aligned}$$

This is 1-dimensional information: information which depends on 1 number, j.

Two dimensional information

Let S be a set, $n \in \mathbb{N}$. Elements of $S^n = \underbrace{S \times S \times \cdots \times S}_{n \text{ copies of } S}$ correspond to sequences $(a_j)_{1..n}$, i.e. functions

$$\begin{aligned} a : \{1, ..., n\} &\to S \\ j &\mapsto a_j. \end{aligned}$$

This is 1-dimensional information: information which depends on 1 number, j.

Elements of $M_n(S)$ correspond to functions

$$\begin{aligned} a : \{1, ..., n\} \times \{1, ..., n\} &\to S \\ (i, j) &\mapsto a_{i,j}. \end{aligned}$$

This is 2-dimensional information: information which depends on 2 numbers, i and j.

Examples

- An image can be described by the colour of each pixel.
  Let C be the set of colours.
  A square 1 megapixel image is an element of $M_{10^3}(C)$.

Examples

- An image can be described by the colour of each pixel.
  Let C be the set of colours.
  A square 1 megapixel image is an element of $M_{10^3}(C)$.
- A relation $R \subseteq \{1, ..., n\} \times \{1, ..., n\}$ can be represented by a matrix $(a_{i,j}) \in M_n(\{0, 1\})$ with

$$a_{i,j} = 1 \iff iRj.$$

Examples

- An image can be described by the colour of each pixel.
  Let C be the set of colours.
  A square 1 megapixel image is an element of $M_{10^3}(C)$.
- A relation $R \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$ can be represented by a matrix $(a_{i,j}) \in M_n(\{0, 1\})$ with

  $$a_{i,j} = 1 \iff iRj.$$

  Example:

$$\begin{tikzcd}
 & 1 \arrow[dd] \arrow[dr] & \\
2 \arrow[loop left] \arrow[rr] & & 3 \arrow[dl] \\
 & 4 \arrow[loop below] &
\end{tikzcd}$$

Examples

- An image can be described by the colour of each pixel.
  Let C be the set of colours.
  A square 1 megapixel image is an element of $M_{10^3}(C)$.

- A relation $R \subseteq \{1, ..., n\} \times \{1, ..., n\}$ can be represented by a matrix $(a_{i,j}) \in M_n(\{0,1\})$ with

$$a_{i,j} = 1 \iff iRj.$$

Example:

$$\begin{tikzcd}
 & 1 \arrow[dd] \arrow[dr] & \\
2 \arrow[loop left] \arrow[rr] & & 3 \arrow[dl] \\
 & 4 \arrow[loop below] &
\end{tikzcd}
\quad \text{is represented by} \quad
\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Another example

- A matrix $(a_{i,j}) \in M_n(\mathbb{Q})$ can define a weighted relation. Let us consider 4 companies, called 1,2,3,4, and let $a_{i,j}$ be the money ($) received by i from j in a year.

Another example

- A matrix $(a_{i,j}) \in M_n(\mathbb{Q})$ can define a weighted relation. Let us consider 4 companies, called 1,2,3,4, and let $a_{i,j}$ be the money ($) received by i from j in a year. Then

$$\begin{pmatrix} 0 & 10^4 & 0 & 10^5 \\ 0 & 0 & 0 & 10^5 \\ 10^4 & 0 & 0 & 10^5 \\ 10^5 & 0 & 0 & 0 \end{pmatrix}$$

represents the situation where :

Another example

- A matrix $(a_{i,j}) \in M_n(\mathbb{Q})$ can define a weighted relation. Let us consider 4 companies, called 1,2,3,4, and let $a_{i,j}$ be the money (\$) received by i from j in a year. Then

$$\begin{pmatrix} 0 & 10^4 & 0 & 10^5 \\ 0 & 0 & 0 & 10^5 \\ 10^4 & 0 & 0 & 10^5 \\ 10^5 & 0 & 0 & 0 \end{pmatrix}$$

represents the situation where :

1 received $\$10^4$ from 2

and $\$10^5$ from 4,

Another example

- A matrix $(a_{i,j}) \in M_n(\mathbb{Q})$ can define a weighted relation. Let us consider 4 companies, called 1,2,3,4, and let $a_{i,j}$ be the money (\$) received by i from j in a year. Then

$$\begin{pmatrix} 0 & 10^4 & 0 & 10^5 \\ 0 & 0 & 0 & 10^5 \\ 10^4 & 0 & 0 & 10^5 \\ 10^5 & 0 & 0 & 0 \end{pmatrix}$$

represents the situation where :

1 received \$$10^4$ from 2

and \$$10^5$ from 4,

2 received \$$10^5$ from 4,

Another example

- A matrix $(a_{i,j}) \in M_n(\mathbb{Q})$ can define a weighted relation. Let us consider 4 companies, called 1,2,3,4, and let $a_{i,j}$ be the money ($) received by i from j in a year. Then

$$\begin{pmatrix} 0 & 10^4 & 0 & 10^5 \\ 0 & 0 & 0 & 10^5 \\ 10^4 & 0 & 0 & 10^5 \\ 10^5 & 0 & 0 & 0 \end{pmatrix}$$

represents the situation where :

1 received $\$10^4$ from 2
and $\$10^5$ from 4,
2 received $\$10^5$ from 4,
3 received $\$10^4$ from 1
and $\$10^5$ from 4,

Another example

- A matrix $(a_{i,j}) \in M_n(\mathbb{Q})$ can define a weighted relation. Let us consider 4 companies, called 1,2,3,4, and let $a_{i,j}$ be the money (\$) received by i from j in a year. Then

$$\begin{pmatrix} 0 & 10^4 & 0 & 10^5 \\ 0 & 0 & 0 & 10^5 \\ 10^4 & 0 & 0 & 10^5 \\ 10^5 & 0 & 0 & 0 \end{pmatrix}$$

represents the situation where :

1 received \$$10^4$ from 2
and \$$10^5$ from 4,
2 received \$$10^5$ from 4,
3 received \$$10^4$ from 1
and \$$10^5$ from 4,
4 received \$$10^5$ from 1.

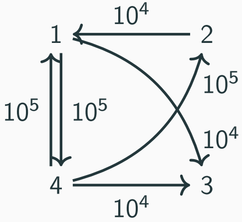
Another example

- A matrix $(a_{i,j}) \in M_n(\mathbb{Q})$ can define a weighted relation. Let us consider 4 companies, called 1,2,3,4, and let $a_{i,j}$ be the money (\$) received by i from j in a year. Then

$$\begin{pmatrix} 0 & 10^4 & 0 & 10^5 \\ 0 & 0 & 0 & 10^5 \\ 10^4 & 0 & 0 & 10^5 \\ 10^5 & 0 & 0 & 0 \end{pmatrix}$$

represents the situation where :

1 received \$$10^4$ from 2
and \$$10^5$ from 4,
2 received \$$10^5$ from 4,
3 received \$$10^4$ from 1
and \$$10^5$ from 4,
4 received \$$10^5$ from 1.



Vectors

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as
an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as
an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**
or as an element of $M_{n\times 1}(\mathbb{Q})$; x is then called a **column vector**.

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as
an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**
or as an element of $M_{n\times 1}(\mathbb{Q})$; x is then called a **column vector**.

The **sum** of two vectors, $\mathrm{x} + \mathrm{y}$, is defined element-wise:

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as

an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**

or as an element of $M_{n\times 1}(\mathbb{Q})$; x is then called a **column vector**.

The **sum** of two vectors, $\mathrm{x} + \mathrm{y}$, is defined element-wise:

$$\mathrm{x} + \mathrm{y} = (x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n).$$

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as

an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**

or as an element of $M_{n\times 1}(\mathbb{Q})$; x is then called a **column vector**.

The **sum** of two vectors, $\mathrm{x} + \mathrm{y}$, is defined element-wise:

$$\mathrm{x} + \mathrm{y} = (x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n).$$

(When viewed as row or column vectors, x and y must be the same shape.)

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as

an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**

or as an element of $M_{n\times 1}(\mathbb{Q})$; x is then called a **column vector**.

The **sum** of two vectors, $\mathrm{x} + \mathrm{y}$, is defined element-wise:

$$\mathrm{x} + \mathrm{y} = (x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n).$$

(When viewed as row or column vectors, x and y must be the same shape.)

There are a number of ways to define the product of two vectors (*e.g* the 'inner' and the 'outer' products) but we will not use them in this course.

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as
an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**
or as an element of $M_{n\times 1}(\mathbb{Q})$; x is then called a **column vector**.

The **sum** of two vectors, $\mathrm{x} + \mathrm{y}$, is defined element-wise:

$$\mathrm{x} + \mathrm{y} = (x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n).$$

(When viewed as row or column vectors, x and y must be the same shape.)

There are a number of ways to define the product of two vectors (*e.g* the 'inner' and the 'outer' products) but we will not use them in this course. However we do need to define the product of a number λ and a vector. In this context the number λ is referred to as a **scalar**, to distinguish it from a vector, and the product $\lambda \mathrm{x}$ is called a **scalar product**.

Vectors and vector arithmetic

For any $n \in \mathbb{N}$ an element $\mathrm{x} = (x_1, ..., x_n) \in \mathbb{Q}^n$ will be called a **vector**.

A vector $\mathrm{x} \in \mathbb{Q}^n$ can be viewed as

an element of $M_{1\times n}(\mathbb{Q})$; x is then called a **row vector**

or as an element of $M_{n\times 1}(\mathbb{Q})$; x is then called a **column vector**.

The **sum** of two vectors, $\mathrm{x} + \mathrm{y}$, is defined element-wise:

$$\mathrm{x} + \mathrm{y} = (x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n).$$

(When viewed as row or column vectors, x and y must be the same shape.)

There are a number of ways to define the product of two vectors (*e.g* the 'inner' and the 'outer' products) but we will not use them in this course. However we do need to define the product of a number λ and a vector. In this context the number λ is referred to as a **scalar**, to distinguish it from a vector, and the product $\lambda\mathrm{x}$ is called a **scalar product**. It is also defined element-wise:

$$\forall\lambda \in \mathbb{Q} \quad \lambda\mathrm{x} = \lambda(x_1, ..., x_n) = (\lambda x_1, ..., \lambda x_n).$$

Examples of vectors and vector arithmetic

- Let $\mathrm{p} = (p_1, p_2, p_3) \in \mathbb{Q}^3$ represent the state of an ecosystem with p_1, p_2, p_3 being the sizes of the populations of three different species.

Examples of vectors and vector arithmetic

- Let $\mathsf{p} = (p_1, p_2, p_3) \in \mathbb{Q}^3$ represent the state of an ecosystem with p_1, p_2, p_3 being the sizes of the populations of three different species. If p_1 increases by 10 individuals, p_2 loses 20 individuals, and p_3 gains 2, then the new state of the ecosystem is

$$\mathsf{p} + \mathsf{c} = (p_1, p_2, p_3) + (10, -20, 2)$$

Examples of vectors and vector arithmetic

- Let $\mathrm{p} = (p_1, p_2, p_3) \in \mathbb{Q}^3$ represent the state of an ecosystem with p_1, p_2, p_3 being the sizes of the populations of three different species. If p_1 increases by 10 individuals, p_2 loses 20 individuals, and p_3 gains 2, then the new state of the ecosystem is

$$\mathrm{p} + \mathrm{c} = (p_1, p_2, p_3) + (10, -20, 2)$$

- Let $\mathrm{a} = (a_1, ..., a_n) \in \mathbb{Q}^n$ represent the amplitudes a_k, $1 \leq k \leq n$, of the harmonic frequencies kf of the fundamental frequency f of a note played on a violin.

Examples of vectors and vector arithmetic

- Let $\mathrm{p} = (p_1, p_2, p_3) \in \mathbb{Q}^3$ represent the state of an ecosystem with p_1, p_2, p_3 being the sizes of the populations of three different species. If p_1 increases by 10 individuals, p_2 loses 20 individuals, and p_3 gains 2, then the new state of the ecosystem is

$$\mathrm{p} + \mathrm{c} = (p_1, p_2, p_3) + (10, -20, 2)$$

- Let $\mathrm{a} = (a_1, ..., a_n) \in \mathbb{Q}^n$ represent the amplitudes a_k, $1 \leq k \leq n$, of the harmonic frequencies kf of the fundamental frequency f of a note played on a violin. Then

$$3\mathrm{a} = 3(a_1, ..., a_n),$$

represents to the same sound, but three times stronger.

Next Lecture

We will learn more about matrix arithmetic and see what type of processes can be modelled using matrices!