HW I

> Definition: Function f : N → N is **space constructible** if there is a f(n)-space
> bounded Turing machine M that for each input of length n it marks exactly f(n)
> cells on the output tape and stops.
>
> Prove that the functions $\log n$, 2^n and $n!$ are space constructible.

$\log n$: Turing machine which computes the function that maps n-length string of 1s to the binary representation of $\log n$ can be constructed as follows.

It counts the number of 1s in its input in binary. That way, we have obtained halved string. Keep doing this until we get 1-length string in the input . Hence $\log_2 n$ can be computed by counting the number of bits in the binary representation of n in the output tape.

We have blanked input of length n and correspondingly we have marked exactly $f(n) = \log_2 n$ cells on the output tape, so such machine exists. Hence space constructible.

2^n : Turing machine which computes the function that maps n-length string of 1s to the binary representation of 2^n can be constructed as follows.

It counts the number of 1s in its input . For each 1 read from the input, mark two cells on the output cell. Keep doing this until input tape is all blanked. Hence 2^n can be computed by counting the number of marked cells in the output tape.

We have blanked input of length n and correspondingly we have marked exactly $f(n) = 2^n$ cells on the output tape, so such machine exists. Hence space constructible.

$n!$: It is space constructible because a machine may take its input from n-length string of 1s to the binary representation of n can be constructed, also we can obtain all values from 1 to n in binary by decrementing the number n by 1. Then output $n!$ can be obtained by using any standard method for multiplying all of the values from 1 to n .

We have blanked input of length n and correspondingly we have marked exactly $f(n) = n!$ cells on the output tape, so such machine exists. Hence space constructible.

> Definition: Function f : N → N, is **time constructible** if some $O(f(n))$ time TM
> M exists that always halt with the binary representation of $f(n)$ on its output
> tape when started on input 1^n.
>
> Prove that the functions $n \log n,\ n\sqrt{n},\ 2^n$ are time constructible.

$n\sqrt{n}$: We need to construct a TM to count the number of 1s in binary, call it M. As the counter of the machine M moves through the input tape, counter increments by 1 for every input position, until it blanks the whole input. This part uses $O(n \log n)$ steps because there are n input positions and each of them takes O(log n) steps, that means we can also say that this machine M bounded by $O(f(n)) = O(n\sqrt{n})$ time.

Then, we compute $n\sqrt{n}$ in binary from the binary representation of n. Any reasonable method of doing so will work in O(n log n) time because the length of the numbers involved is O(log n).

There exists a (multi-tape deterministic) Turing machine that for n given, halts in O(f(n)) time , hence time constructible.

> Prove that the language $L = \{a^k b^{2^k} : k > 0\}$ is of space complexity log n.

- Tape 1: input
- Tape 2: counter 1,
  keeping the position of the left symbol to be compared with the other on the right symbols.
- Tape 3: counter 2,
  If it is empty, copy counter 1 and increase it by 1.
- Tape 4: copy counter 1, then by decrementing it find the
  position of the left symbol, check if it equals to ‘a’, now copy counter 2,
  by decrementing it check all of the cells from the right equals to ‘b’ until tape 4 is emptied ; if OK then increment counter 1, double counter 2 by incrementing it by number of its size and repeat, check the next cells.
- We need storage only for the counters and we need logarithmic space.

> Prove that the language $L = \{a^k b^{2^k} : k > 0\}$ can be recognized by a one tape TM in time O(n log n). Be careful of words that do not belong to L.

Single tape:
L : let w be a given input

1. Scan across the tape and reject if a ‘b’ is found to the right of ‘a’.
2. Repeat if both 0s and 1s remain on the tape:
3. Scan across the tape, crossing off a single ‘a’ and ‘b’s as the same number of ‘b’s crossed before. If no ‘b’ had been crossed before, cross two ‘b’s.
4. If ‘a’s still remain after all the ‘b’s have been crossed off, or if ‘b’s
   still remain after all the ‘a’s have been crossed off, reject. Otherwise, if neither ‘a’s nor ‘b’s remain on the tape, accept.

Now, for example $M = \{a^k b^k : k > 0\}$ uses linear time.

But in this example we also need logarithmic time for crossing the ‘b’s.
Because each time we cross an ‘a’ we need to cross more ‘b’ and it increases logarithmically.
Hence O(n.logn).

> Let M be a NTM of time complexity T(n). Prove that $L(M) \in DTIME(c^{T(n)})$, for some c > 0.

As we know how to simulate NTM by a TM:
We can simulate this M by a deterministic Turing machine N. If we assumed that at each non-final configuration there are exactly 2 possibilities for the next step, we would use $2^{T(n)}$ time. Hence we are safe to say that language M is bounded by $O(c^{T(n)})$ for some c>0. Hence $L(M) \in DTIME(c^{T(n)})$.