- Aim: reduce input size by half in a scan. (so that we have to repeat it only log n times).

<u>Algorithm:</u> - If a 0 appears after 1, reject.

Repeat: - If total no. of symbols odd, reject.
- Cross off alternative 0's. Then 1's.

- No symbols left → Accept. Else Reject.

- Complexity "between models":

<u>Thm:</u> If $t(n) \geq n$, then
every $t(n)$ multitape turing machine
has an equivalent $O(t^2(n))$-time
single-tape turing machine.

Proof outline:

- Simulate the k-tape turing machine on a single-tape machine, as we did earlier.
- For one move on k-tape machine, this TM makes 2 scans from left to right: one to 'decide' the moves, second to 'update' the symbols.

- i.e., for one step (move) on k-tape TM, it takes O(length of the tape) on new TM.
- So, what is length of this tape, in the worst case? There are k 'parts'.
- Each part has length at most $t(n)$.
- Hence total length $= k \times t(n) = O(t(n))$.
- $t(n)$ such moves $\rightarrow$ Max $O(t^2(n))$ moves.

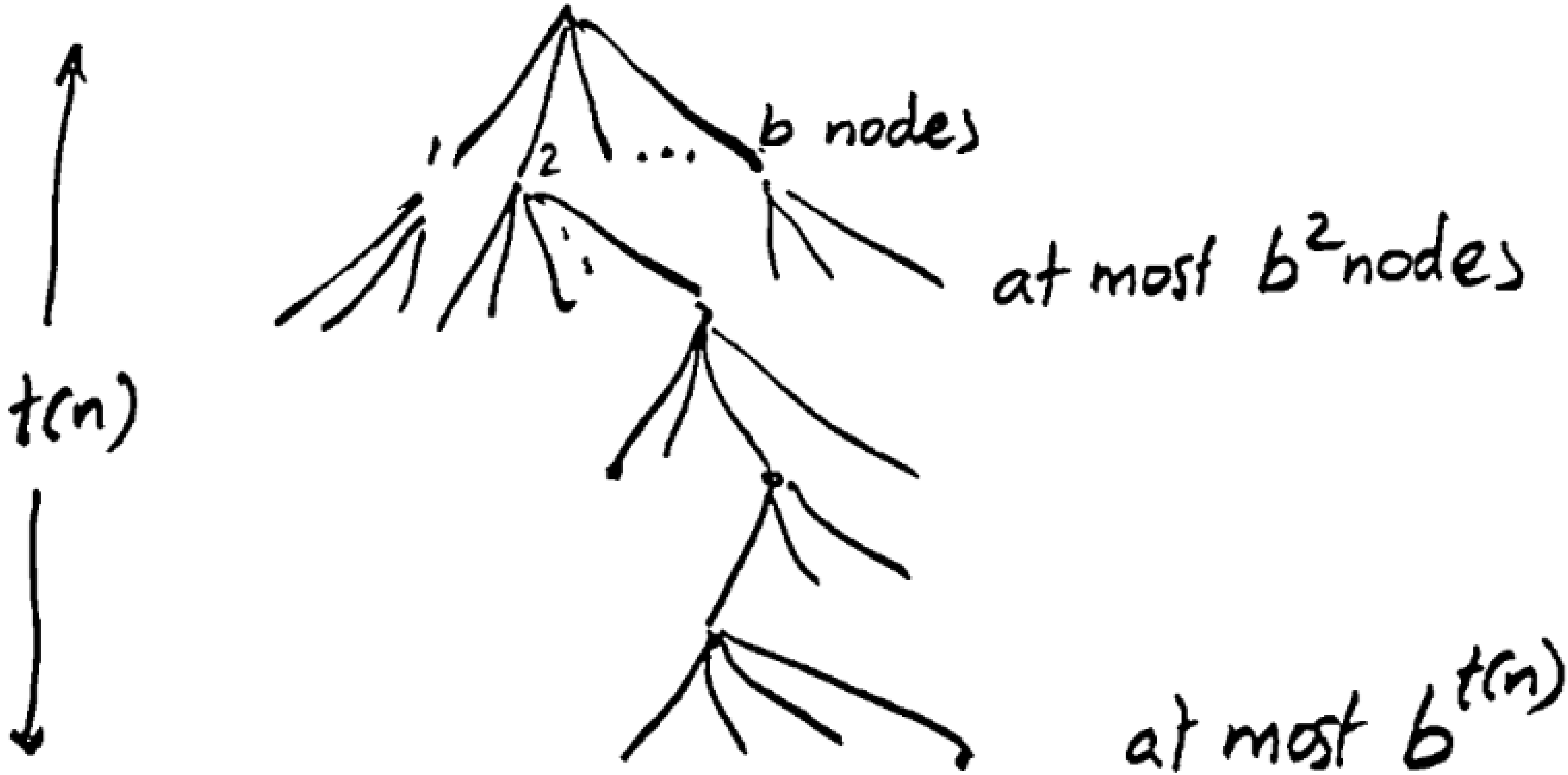
· Non-deterministic to deterministic:

<u>Thm</u>: If $t(n) \geq n$, then

every $t(n)$ time non-deterministic single-tape turing machine has an equivalent $2^{O(t(n))}$ time deterministic turing machine.

Proof idea:

- Simulate the non-deterministic TM N on a deterministic TM D, as earlier.
- Let b be the maximum 'branching'. (Max. no. of possible moves at a point).
- 'Height' of the computation tree $= t(n)$. Total no. of leaves at most $b^{t(n)}$.


b nodes
1
2
...
at most b^2 nodes
$t(n)$
at most $b^{t(n)}$

- While executing the moves on D, it traverses this tree breadth-first.
- By the time it reaches level $t(n)$, it has traversed all nodes before that level.
- No. of nodes $= O(b^{t(n)})$.
- Running time $\leq t(n)\, b^{t(n)} = 2^{O(t(n))}$.
- Conversion to single tape only squares it.

• Class P:

TIME($t(n)$): Set of languages /problems that are decided by $O(t(n))$ time single tape turing machine.

TIME(n), TIME(n^2), TIME($n \log n$) etc.

$$P = \bigcup_k \text{TIME}(n^k).$$