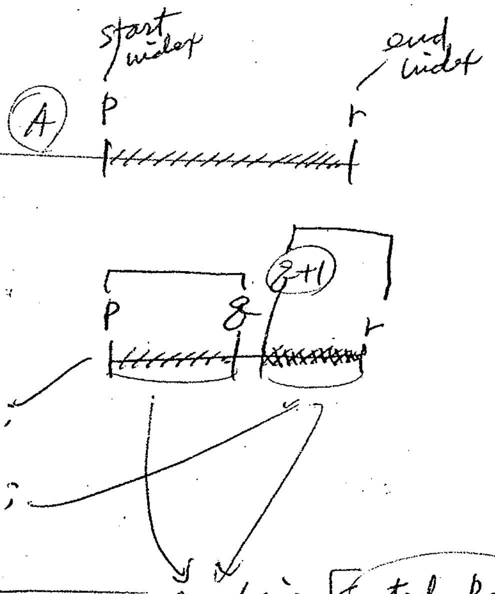
algo

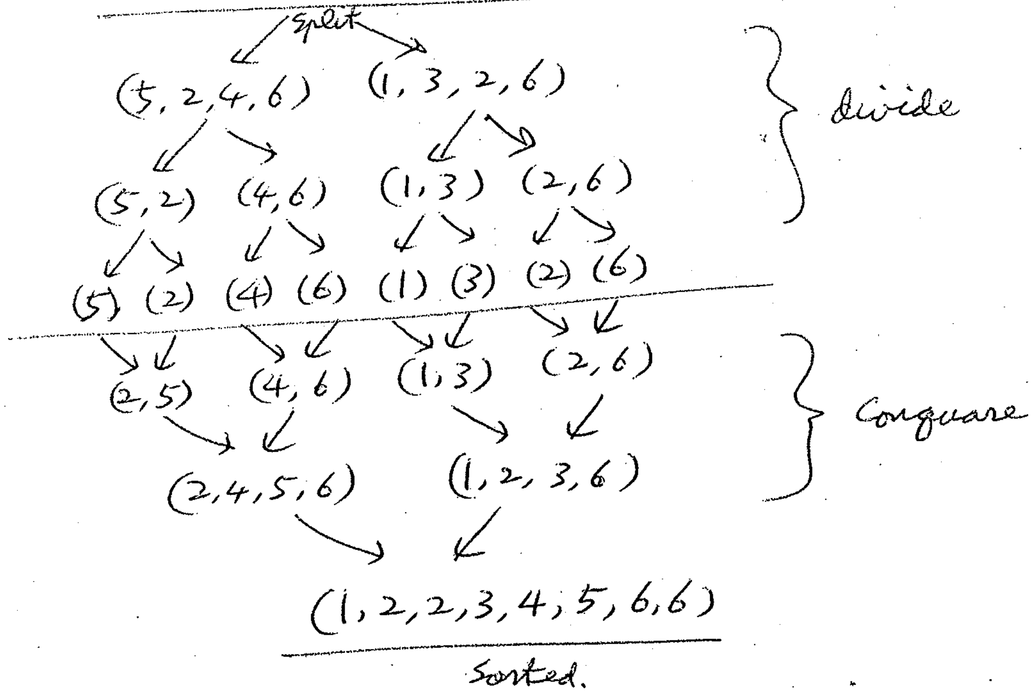
```
mergeSort(A, p, r)
{ if (p<r)
      $q \leftarrow \lfloor (p+r)/2 \rfloor$;
      mergeSort(A, p, q);
      mergeSort(A, q+1, r);
      merge(A, p, q, r);
}
```



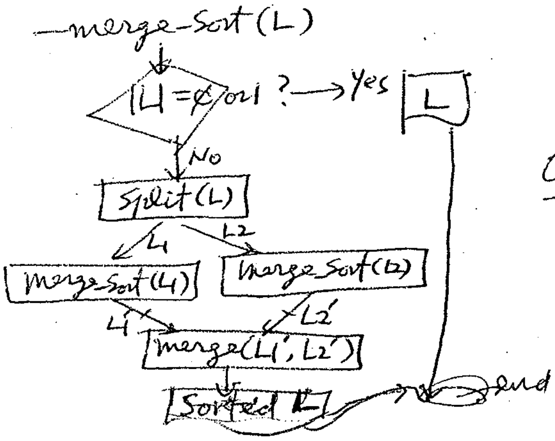
Combine sorted p~q and
sorted q+1~r

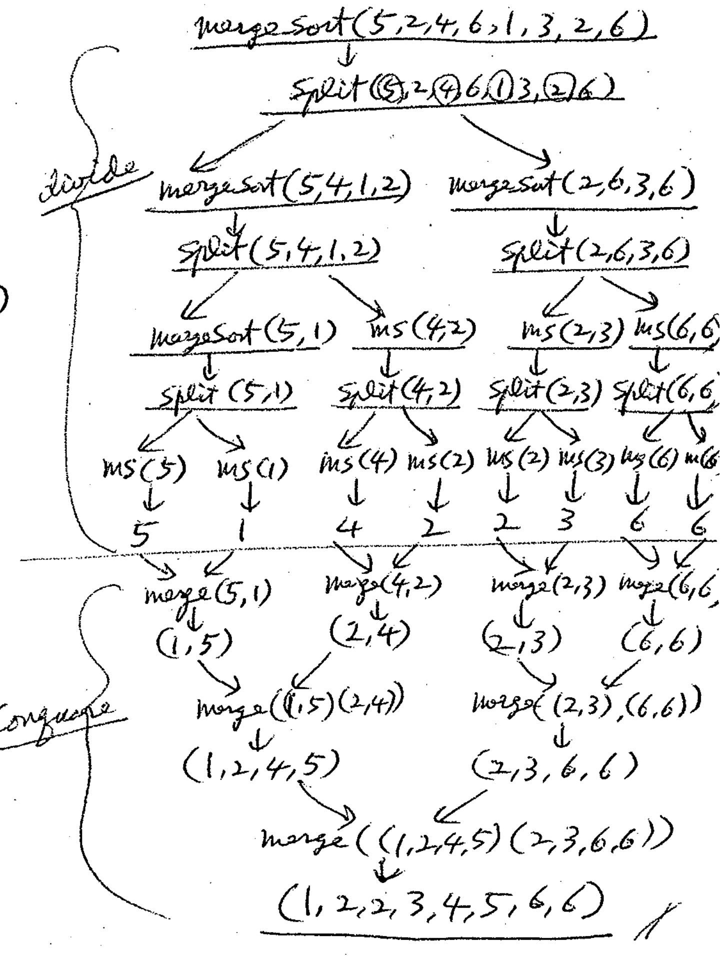
mergesort in Scheme

ex) (5. 2, 4, 6, 1, 3, 2, 6)



```
— mergeSort(L)
{ if(|L| ≥ 2)
    ( Split(L) → L1
               → L2
      merge-Sort (L1); ⇒ L1' (Sorted L1)
      merge-Sort (L2); ⇒ L2' (Sorted L2)
      merge (L1', L2'); ⇒ L' (Sorted L)
      return L';
  else
      return L;
}
```





Pseudo-code

```
— merge(X, Y)
  { if (X is null), return Y;
    else if (Y is null), return X;
    else if (1st-ele. of X ≤ 1st-ele. of Y)
            construct (1st-ele. of X, merge(○, ○))
              cons                             X without 1st-ele.   Y
         else
            construct (1st-ele. of Y, merge(○, ○))
                                             X    Y without 1st ele.
  }


— merge_sort(X)
  { if (X is null), return X;  —— () ⇒ ()
    else if (X is 1-ele. list), return X;  —— (1) ⇒ (1).
    else
    return merge( merge_sort(○), merge_sort(○))
  }
                 1st part from Split(X)    2nd part from Split(X)
                       ||                        ||
                 (car (split x))           (cdr (split x))

                                ((1 2) 3 4)

— Split(x) - - (1 3 2 4) ⇒ ((1 2) 3 4)
  { if (X is null), construct (()); —— (cons '() '())
    else if (X is 1-ele. list), construct ((1));   (1) → ((1))
    else ( ---- ) ———— (1 3 2 4) ⇒ ((1 2) 3 4)  ) either
                              or ⇒ ((1 3) 2 4)  ) is ok.
```