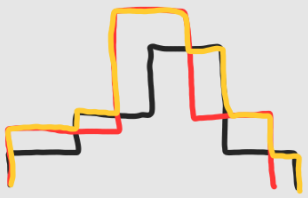
3)

```
CountingSortPlus(A, N, K)
    aloca vetor C[1...K]
    para i <- 1 até N faça
        C[A[i]] <- C[A[i]] + 1
    para i <- 2 até N faça
        C[i] <- C[i] + C[i-1]
    return C.
```

Com o pré-processamento acima, que exige $O(N+K)$, obtemos um vetor tal que cada índice representa um valor da coleção A e cada posição do vetor representa a quantidade de elementos menores ou iguais ao índice. Portanto, dado um intervalo $[a, b]$, basta que seja feito $C[b] - C[a-1]$.

a)



```
Intercala (S1, S2, N1, N2)
  i = 2, y = 2, K = 2
  S3[1... N1+N2]
  S3[1] = 0
  enquanto i ≤ N1 ou y ≤ N2
     S3[K] = max (S1[i], S2[y])
     se S1[i+1] > S2[y+1]
     então S3[K+1] = S2[y+1], y += 2
     senao se S1[i+1] == S2[y+1]          // caso as bases sejam iguais
           então  S3[k+1] = S1[i+1], i += 2, y += 2
           senão  S3[k+1] = S1[i+1], i += 2
  enquanto i ≤ N1                         // preenche o que resta
     S3[K] = S1[i], S3[k+1] = S1[i+1], i += 2
  enquanto y ≤ N2                         // preenche o que resta
     S3[K] = S2[y], S3[k+1] = S2[y+1], y += 2
  S4[1... N1+N2], S4[1] = 0, K = 2, i = 2
  enquanto K < N1+N2                      // corrige repetição de alturas
     se k+2 < N1+N2 e S3[K] == S3[K+2]
     então S4[i] = S3[K+2], S4[K+3], i += 2, K += 2
     senão S4[i] = S3[K], S4[i+1] = S3[K+1], i += 2, K += 2
  return S4
```

O algoritmo unifica os skylines porque usa dois apontadores e prioriza aquele que tem a maior altura e menor base tal que no fim da execução ambos apontam para o fim dos vetores. Além disso, ele é O(N) porque percorre cada posição uma única vez

b)

```
SKyline(S, p, r)
    se p < r
        q = ⌊(p+r)/2⌋
        S1 = SKyline(S[p...q], p, q)
        S2 = SKyline(S[q+1...r], q, r)
        SK = Intercala(S1, S2, q-p+1, r-q+1)
        return SK
    se r - p == 1
        return transforma(S)
```

```
transforma(S): recebe um prédio e retorna sua skyline
    r = S.r
    l = S.l
    h = S.h
    se r == 0
        return (0, h, l)
    return (0, 0, r, h, l)
```

A ideia do algoritmo consiste na divisão e conquista vista no Mergesort porque o skyline de uma coleção com apenas uma silhueta de prédio é o transforma(prédio). Assim, usamos o algoritmo anterior para fazer a conquista após a divisão.

Dado o comportamento verossímil ao mergesort, verifica-se que é $O(n \log n)$ porque passamos por cada porção uma única vez e sempre dividimos o conjunto de silhuetas pela metade.