# Wykład 5_2
# Algorytm ograniczania liczby serii za pomocą kopcowego rozdzielania serii początkowych

## Założenia:

**1.** Pamięć wewnętrzna ma ograniczone rozmiary

**2.** Pamięć zewnętrzna ma rozmiary „nieograniczone”

**3.** Czas dostępu do danych w pamięci wewnętrznej jest niezależny od położenia danych (np. dostęp indeksowany w tablicach)

**4.** Czas dostępu do danej w pamięci wewnętrznej jest dużo mniejszy od czasu dostępu do danej w pamięci zewnętrznej, stąd jest pomijany w szacowaniu wydajności algorytmów zewnętrznych

**5.** Bezpośrednio po zapisie lub odczycie danej w pamięci zewnętrznej dostęp do niej jest sekwencyjny i niesekwencyjny - zarówno do odczytu i zapisu

**6.** Czas dostępu do danych w pamięci zewnętrznej jest zależny od położenia- zaleca się sekwencyjne czytanie i zapis danych, gdyż koszt dostępu niesekwencyjnego jest dużo wyższy od sekwencyjnego

***Przykład***
Zawartość pliku źródłowego złożonego z 20 elementów

| -1 | -4 | 0 | 5 | 7 | 4 | -4 | 8 | -1 | 5 | 9 | 2 | 7 | 4 | 7 | 9 | -5 | -2 | -5 | -6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

1) utworzenie kopca (m=14)

| *indeksy* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *14* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *wartości* | -4 | -1 | -4 | -1 | 5 | 2 | 0 | 8 | 5 | 7 | 9 | 4 | 7 | 4 |

2) przepuszczanie przez kopiec elementów z pliku, należących do tej samej serii (większych od elementu pierwszego);

| | *indeksy* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *14* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -4 ← | | -4 | -1 | -4 | -1 | 5 | 2 | 0 | 8 | 5 | 7 | 9 | 4 | 7 | 4 | ← 7 |
| | | 7 | -1 | -4 | -1 | 5 | 2 | 0 | 8 | 5 | 7 | 9 | 4 | 7 | 4 | |
| -4 ← | | -4 | -1 | 0 | -1 | 5 | 2 | 4 | 8 | 5 | 7 | 9 | 4 | 7 | 7 | ← 9 |
| | | 9 | -1 | 0 | -1 | 5 | 2 | 4 | 8 | 5 | 7 | 9 | 4 | 7 | 7 | |
| | | -1 | -1 | 0 | 5 | 5 | 2 | 4 | 8 | 9 | 7 | 9 | 4 | 7 | 7 | |

3) po rozpoczęciu nowej serii usuwanie po kolei elementów z dolnego kopca i stopniowe zapełnianie górnej części tablicy (jeśli korzeń-ojciec górnego kopca będzie mniejszy od połowy tablicy - należy po każdym wstawieniu odtwarzać kopiec w górnej części tablicy; w przykładzie ten przypadek nie wystąpił);

| | *indeksy* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *14* | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 ← | *1..14* | -1 | -1 | 0 | 5 | 5 | 2 | 4 | 8 | 9 | 7 | 9 | 4 | 7 | 7 | ← -5 |
| | *1..13* | 7 | -1 | 0 | 5 | 5 | 2 | 4 | 8 | 9 | 7 | 9 | 4 | 7 | | |
| -1 ← | *1..13* | -1 | 5 | 0 | 7 | 5 | 2 | 4 | 8 | 9 | 7 | 9 | 4 | 7 | *-5* | ← -2 |
| | *1..12* | 7 | 5 | 0 | 7 | 5 | 2 | 4 | 8 | 9 | 7 | 9 | 4 | | *-5* | |
| 0 ← | *1..12* | 0 | 5 | 2 | 7 | 5 | 4 | 4 | 8 | 9 | 7 | 9 | 7 | *-2* | *-5* | ← -5 |
| | *1..11* | 7 | 5 | 2 | 7 | 5 | 4 | 4 | 8 | 9 | 7 | 9 | | *-2* | *-5* | |
| 2 ← | *1..11* | 2 | 5 | 4 | 7 | 5 | 7 | 4 | 8 | 9 | 7 | 9 | *-5* | *-2* | *-5* | ← -6 |
| | *1..10* | 9 | 5 | 4 | 7 | 5 | 7 | 4 | 8 | 9 | 7 | | *-5* | *-2* | *-5* | |
| | *1..10* | 4 | 5 | 4 | 7 | 5 | 7 | 9 | 8 | 9 | 7 | *-6* | *-5* | *-2* | *-5* | koniec pliku |

4) po wyczerpaniu pliku źródłowego usuwanie po kolei elementów z dolnego kopca i przesuwanie elementów w górnej części tablicy;

| | *indeksy* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *14* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 ← | *1..10* | 4 | 5 | 4 | 7 | 5 | 7 | 9 | 8 | 9 | 7 | *-6* | *-5* | *-2* | *-5* |
| | *1..9* | 7 | 5 | 4 | 7 | 5 | 7 | 9 | 8 | 9 | | *-6* | *-5* | *-2* | *-5* |
| 4 ← | *1..9* | 4 | 5 | 7 | 7 | 5 | 7 | 9 | 8 | 9 | *-5* | *-6* | *-5* | *-2* | |
| | *1..8* | 9 | 5 | 7 | 7 | 5 | 7 | 9 | 8 | | *-5* | *-6* | *-5* | *-2* | |
| 5 ← | *1..8* | 5 | 5 | 7 | 7 | 9 | 7 | 9 | 8 | *-2* | *-5* | *-6* | *-5* | | |
| | *1..7* | 8 | 5 | 7 | 7 | 9 | 7 | 9 | | *-2* | *-5* | *-6* | *-5* | | |
| 5 ← | *1..7* | 5 | 7 | 7 | 8 | 9 | 7 | 9 | *-5* | *-2* | *-5* | *-6* | | | |
| | *1..6* | 9 | 7 | 7 | 8 | 9 | 7 | | *-5* | *-2* | *-5* | *-6* | | | |
| 7 ← | *1..6* | 7 | 8 | 7 | 9 | 9 | 7 | *-6* | *-5* | *-2* | *-5* | | | | |
| | *1..5* | 7 | 8 | 7 | 9 | 9 | | *-6* | *-5* | *-2* | *-5* | | | | |
| 7 ← | *1..5* | 7 | 8 | 7 | 9 | 9 | *-5* | *-6* | *-5* | *-2* | | | | | |
| | *1..4* | 9 | 8 | 7 | 9 | | *-5* | *-6* | *-5* | *-2* | | | | | |
| 7 ← | *1..4* | 7 | 8 | 9 | 9 | *-2* | *-5* | *-6* | *-5* | | | | | | |
| | *1..3* | 9 | 8 | 9 | | *-2* | *-5* | *-6* | *-5* | | | | | | |
| 8 ← | *1..3* | 8 | 9 | 9 | *-5* | *-2* | *-5* | *-6* | | | | | | | |
| | *1..2* | 9 | 9 | | *-5* | *-2* | *-5* | *-6* | | | | | | | |
| 9 ← | *1..2* | 9 | 9 | ***-6*** | *-5* | *-2* | ***-5*** | | | | | | | | |
| 9 ← | *1..2* | 9 | 9 | *-6* | *-5* | *-2* | *-5* | | | | | | | | |
| | *1..1* | 9 | | *-6* | *-5* | *-2* | *-5* | | | | | | | | |
| 9 ← | *1..1* | 9 | ***-5*** | *-6* | ***-5*** | ***-2*** | | | | | | | | | |
| 9 ← | *1..1* | 9 | *-5* | *-6* | *-5* | *-2* | | | | | | | | | |
| 9 ← | *1..1* | | *-5* | *-6* | *-5* | *-2* | | | | | | | | | |
| | *1..4* | ***-2*** | ***-5*** | ***-6*** | *-5* | | | | | | | | | | |
| | *1..4* | *-6* | *-5* | ***-2*** | *-5* | | | | | | | | | | |

5) wykonanie górnego kopca i usuwanie po kolei elementów z górnego kopca;

| | *indeksy* | *1* | *2* | *3* | *4* |
|---|---|---|---|---|---|
| -6 ← | *1..4* | -6 | -5 | -2 | -5 |
| | *1..3* | **-5** | **-5** | **-2** | |
| -5 ← | *1..3* | -5 | -5 | -2 | |
| | *1..2* | **-2** | **-5** | | |
| -5 ← | *1..2* | -5 | -2 | | |
| | *1..1* | -2 | | | |
| -2 ← | *1..1* | -2 | | | |

Plik wyjściowy

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -4 | -4 | -1 | -1 | 0 | 2 | 4 | 4 | 5 | 5 | 7 | 7 | 7 | 8 | 9 | 9 | -6 | -5 | -5 | -2 |

## Algorytm rozdzielania serii początkowych - poziom konceptualny

***(1)*** *Wczytaj pierwsze elementy z pliku źródłowego do tablicy, utwórz kopiec i ustaw licznik serii. Jeżeli rozmiar pliku jest niewiększy od rozmiaru kopca przejdź do kroku* ***(3)****.*

***(2)*** *Dopóki nie wyczerpiesz pliku źródłowego, wykonuj:*

- ***(2.1)*** *Zapisz do pliku wyjściowego najmniejszy element z korzenia dolnego kopca*
- ***(2.2)*** *Odczytaj kolejny element z pliku źródłowego; jeżeli ten element należy do bieżącej serii, umieść go w korzeniu dolnego kopca i odtwórz dolny kopiec, w przeciwnym przypadku:*
- ***(2.3)*** *Jeżeli kolejny element odczytany z pliku źródłowego należy do nowej serii, wtedy umieść ostatni element kopca w korzeniu dolnego kopca, zmniejsz rozmiar dolnego kopca o 1 i umieść w korzeniu górnego kopca (poprzedni koniec dolnego kopca) odczytany element:*
  - ***(2.3.1)*** *jeżeli korzeń górnego kopca znajduje się w pierwszej połowie całej tablicy, odtwórz górny kopiec;*
  - ***(2.3.2)*** *jeżeli opróżnisz dolny kopiec, zwiększ liczbę serii o 1; kopiec górny wypełnia całą tablicę;*

***(3)*** *Dopóki tablica nie jest wyczerpana, wykonuj:*

- ***(3.1)*** *Zapisz w pliku element z korzenia dolnego kopca, przestaw element z końca kopca na początek, zmniejsz rozmiar dolnego kopca o 1 i odtwórz dolny kopiec;*
- ***(3.2)*** *Jeśli istnieje górny kopiec, przestaw ostatni element w górnym kopcu na jego korzeń (pozycja zwolniona przez ostatni element dolnego kopca) i zmniejsz o 1 indeksy górny i dolny górnego kopca - jeżeli indeks korzenia górnego kopca znajduje w pierwszej połowie całej tablicy, odtwórz górny kopiec;*

***(4)*** *Jeśli wyczerpano dolny kopiec, a istnieje górny kopiec, należy opróżnić go zwiększając liczbę serii o 1, jeśli chociaż raz nastąpiła zmiana serii (wykonanie kroku 2.3) .*

## Implementacja algorytmu w języku C/C++

```
// Rozdzielanie_serii_poczatkowych;
//Przygotowanie pliku do wydajnego sortowania zewnętrznego
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

const int m=14;
const int mh=m/2;
const char nazwa[] = "plik0";
struct obiekt
 {  int klucz;
 };
obiekt kopiec[m];

inline void zamien(obiekt &a, obiekt &b);
inline long lewy(long ociec);
inline long prawy(long ociec);
void zbuduj_kopiec(obiekt t[], long l, long p);
void przywroc_kopiec(obiekt t[], long p, long ojciec);

void wybierz_serie(long&);
void rozdzielanie_serii(const char nazwa[],obiekt kopiec[]);

void wygeneruj_losowo_zawartosc_pliku(const char nazwa[]);
void wydruk(const char nazwa[]);
//---------------------------------------------------------------------

void main(int argc, char* argv[])
{
  clrscr();
  wygeneruj_losowo_zawartosc_pliku(nazwa);    //generuj losowo plik plik0
  wydruk(nazwa);
  rozdzielanie_serii(nazwa,kopiec);                  //rozdziel serie
  wydruk(nazwa);
 }
```

```
void wygeneruj_losowo_zawartosc_pliku(const char nazwa[])
 {  obiekt buf;   //element pliku
    long dl, los; //dl-rozmiar pliku, los-uzywane przy generacji pliku
    FILE* plik;

    plik=fopen(nazwa,"wb");
    dl=50; los=77;  los=(131*los) % 2147;
    do
     { los=(131*los)%2172;
       buf.klucz=los / 217;
       fwrite(&buf, sizeof(obiekt),1,plik);
       dl--;
     } while(dl!=0);
    fclose(plik);
 }

void wydruk(const char nazwa[])
 {long z;
  obiekt buf;
  FILE* plik;
  plik = fopen (nazwa,"rb");     //jesli nie mozna otworzyc pliku,
  if (plik==NULL) exit(1);    //nalezy przerwac program
  printf("%s\n",nazwa);
  z=0;
  while (fread(&buf, sizeof(buf),1,plik)==1)  //jesli nie osoiagniwto konca pliku
   { printf("%5d",buf.klucz);
     z++;
     if (z % 300 == 0)
       if (getch()=='k') break;}
   printf(" koniec\n");
   getch();
   fclose(plik);
  }

inline void zamien(obiekt  &a, obiekt  &b)
   { obiekt pom=a;
     a=b;
     b=pom;}
```

```
inline long lewy(long ojciec )
  { return ojciec * 2+1;}

inline long prawy(long ojciec)
  { return ojciec * 2 + 2;}

void przywroc_kopiec(obiekt t[], long p, long ojciec)
 {long ll, pp, min;
  while (ojciec < p)
   { ll = lewy(ojciec);
     pp = prawy(ojciec);
     if (ll <= p && t[ll].klucz < t[ojciec].klucz) min = ll;
     else min = ojciec;
     if (pp <= p && t[pp].klucz < t[min].klucz) min = pp;
     if (min != ojciec)
       {  zamien(t[min], t[ojciec]);
          ojciec=min;}
     else break;
  }
}

 void zbuduj_kopiec(obiekt t[], long l, long p)
 { for  (long i = (l+p)/2; i>=l; i--)
       przywroc_kopiec(t, p, i); }

 void wybierz_serie(long& licznik)
  {  licznik++; }

 long filesize(FILE *plik)
  {  long bpozycja, rozmiar;

     bpozycja = ftell(plik);
     fseek(plik, 0L, SEEK_END);
     rozmiar = ftell(plik);
     fseek(plik, bpozycja, SEEK_SET);
     return rozmiar;
  }
```

```
void rozdzielanie_serii(const char nazwa[], obiekt kopiec[])
 { int g,p, ile, nowa_seria;
   long licznik, rozmiar; //licznik serii, rozmiar= liczba elementów w pliku
   obiekt buf;
   FILE *plik0, *plik;
                                  //utwórz serie początkowe przez kopcowanie
   plik0= fopen (nazwa,"rb");
   if (plik0==NULL) exit(1);
   licznik=0;
   plik = fopen ("plikrrp","wb");
   rozmiar= filesize(plik0);
   wybierz_serie(licznik);
   nowa_seria= 0;
              //krok 1. Wypełnij tablice kopiec
   ile = fread(kopiec, sizeof(obiekt),m,plik0);
              //krok 2. Budowa kopca o indeksach 0..ile-1
   zbuduj_kopiec(kopiec, 0, ile-1);
   p= ile-1;
   if (rozmiar > m*2)
    {         //krok 3. Przechodzenie przez zapełniony kopiec
      while (fread(&buf, sizeof(obiekt),1,plik0)==1)
       { fwrite(&kopiec[0], sizeof(obiekt),1,plik);
         if (kopiec[0].klucz <= buf.klucz)     //nowy obiekt należy do tej samej serii
          { kopiec[0]= buf;
            przywroc_kopiec(kopiec, p, 0); }    //odtwórz dolny kopiec o ind. 0..p
         else
          { nowa_seria= 1; //nowy obiekt należy do następnej serii
            kopiec[0]= kopiec[p];
            przywroc_kopiec(kopiec, p-1, 0); //odtwórz dolny kopiec 0..p-1
            kopiec[p]= buf;
              //odtwórz górny kopiec w podtablicy o indeksach p..m, gdy korzeń-ojciec p tego
              //kopca znajduje się w pierwszej połowie całej tablicy
          if (p<= mh-1) przywroc_kopiec(kopiec, m-1, p);
            p--;
             if (p==-1)
            { p=m-1; wybierz_serie(licznik);} //kopiec jest pełen, zacznij nowa serie
          }
       }
    }                                                  //koniec zapełniania kopców
```

```
g = ile-1;                  //zapamiętaj rozmiary górnego kopca

      //krok.4 Wypisz dolny kopiec
do
 { fwrite(&kopiec[0], sizeof(obiekt),1,plik);
   kopiec[0]= kopiec[p];
   przywroc_kopiec(kopiec, p-1, 0);    //odtwórz dolny kopiec 0..p-1
   kopiec[p]= kopiec[g];               //przestaw ostatni element górnego kopca
                                         //na początek tego kopca
   g--;                                //i zmniejsz rozmiar górnego kopca p..g
   if (p <= mh-1) przywroc_kopiec(kopiec, g, p);
          //odtwórz górny kopiec p..g,
          //gdy korzeń-ojciec p tego kopca znajduje się w pierwszej połowie całej tablicy
   p--;                                //i zmniejsz rozmiar dolnego kopca
 } while (p>=0);

      //krok 5. Wypisz górny kopiec 0..g, generuj ostatnia serie
 if (nowa_seria)
       wybierz_serie(licznik);
 while (g >= 0)
  { fwrite(&kopiec[0], sizeof(obiekt),1,plik);
    kopiec[0]= kopiec[g];
    g--;
    przywroc_kopiec(kopiec, g, 0);          //przywróć kopiec 0..g
  }
  printf("licznik=%i\n",licznik);
  fclose(plik0); remove(nazwa);
  fclose(plik);
  if (rename("plikrrp", nazwa) == 0)
     printf("zmiana nazwy pliku %s to %s.\n", "plikrrp", nazwa);
  else
     perror("zmiana nazwy");}
```

## Podsumowanie

1. Zakłada się, że w ciągu losowo rozłożonych kluczy spodziewana długość serii równa się ***2***.
2. W ciągu losowo rozłożonych kluczy spodziewana długość serii równa się ***2,*** natomiast po przejściu przez kopiec o rozmiarze ***m,*** wynosi ***2m*** na podstawie analizy probabilistycznej. Stąd współczynnik usprawnienia wynosi ***m***.

**Wniosek**: Efektywne rozwiązanie sortowania dużych plików można rozwiązać przez przygotowanie serii początkowych metodą **rozdzielania serii przez kopcowanie** (współczynnik zmniejszenia liczby serii w pliku jest równy rozmiarowi kopca czyli maksymalnej liczbie elementów tablicy reprezentującej kopiec);